SBRC 2019

Gramado | RS

XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

6 a 10 de maio

Gramado | RS

Minicursos do SBRC 2019

sbrc2019.sbc.org.br

XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

6 a 10 de maio

Gramado | RS

Minicursos do SBRC 2019

sbrc2019.sbc.org.br

EDITORA
Sociedade Brasileira de Computação (SBC)

ORGANIZAÇÃO
Alberto Egon Schaeffer Filho (UFRGS)
Weverton Luis da Costa Cordeiro (UFRGS)
Miguel Elias Mitre Campista (UFRJ)

REALIZAÇÃO
Sociedade Brasileira de Computação (SBC)
Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)
Laboratório Nacional de Redes de Computadores (LARC)

Copyright ©2019 da Sociedade Brasileira de Computação
Todos os direitos reservados

Capa: Dórian Fogliatto
Produção Editorial: Carlos Raniery P. dos Santos (UFSM)

Cópias Adicionais:
Sociedade Brasileira de Computação (SBC)
Av. Bento Gonçalves, 9500- Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia - CEP 91.509-900 - Porto Alegre - RS
Fone: (51) 3308-6835
E-mail: sbc@sbc.org.br

Dados Internacionais de Catalogação na Publicação (CIP)

S612 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (37. : 2019 : Gramado, RS)
Minicursos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos [recurso eletrônico] – organizadores: Alberto Egon Schaeffer Filho, Weverton Luis da Costa Cordeiro, Miguel Elias Mitre Campista – Porto Alegre : SBC, 2021.

ISBN 978-65-87003-55-9

1.Redes de computadores. 2. Sistemas distribuídos. I. Schaeffer Filho, Alberto Egon. II. Cordeiro, Weverton Luis da Costa. III. Campista, Miguel Elias Mitre. IV. Título.

CDU 004

Catalogação elaborada por Francine Conde Cabral
CRB-10/2606

Sociedade Brasileira de Computação

Presidência
Lisandro Zambenedetti Granville (UFRGS), Presidente
Thais Vasconcelos Batista (UFRN),Vice-Presidente

Diretorias
Renata de Matos Galante (UFGRS), Diretora Administrativa
Carlos André Guimarães Ferraz (UFPE), Diretor de Finanças
Antônio Jorge Gomes Abelém (UFPA), Diretor de Eventos e Comissões Especiais
Renata Mendes de Araujo (UPM), Diretora de Educação
José Viterbo Filho (UFF), Diretor de Publicações
Claudia Lage Rebello da Motta (UFRJ), Diretora de Planejamento e Programas Especiais
Marcelo Duduchi Feitosa (CEETEPS), Diretor de Secretarias Regionais
Eliana Almeida (UFAL), Diretora de Divulgação e Marketing
Ricardo de Oliveira Anido (UNICAMP), Diretor de Relações Profissionais
Esther Colombini (UNICAMP), Diretora de Competições Científicas
Raimundo José de Araújo Macêdo (UFBA), Diretor de Cooperação com Sociedades Científicas
Cláudia Cappeli (UNIRIO), Diretora de Articulação com Empresas

Contato
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS
CNPJ: 29.532.264/0001-78
http://www.sbc.org.br

Laboratório Nacional de Redes de Computadores (LARC)

Diretor do Conselho Técnico-Científico
Paulo André da Silva Gonçalves (UFPE)

Vice-Diretora do Conselho Técnico-Científico
Rossana Maria de Castro Andrade (UFC)

Diretor Executivo
Ronaldo Alves Ferreira (UFMS)

Vice-Diretor Executivo
Danielo Gonçalves Gomes (UFC)

Membros Institucionais
SESU/MEC, INPE/MCT, UFRGS, UFMG, UFPE, UFCG (ex-UFPB Campus Campina Grande), UFRJ, USP, PUC-Rio, UNICAMP, LNCC, IME, UFSC, UTFPR, UFC, UFF, UFSCar, IFCE (CEFET-CE), UFRN, UFES, UFBA, UNIFACS, UECE, UFPR, UFPA, UFAM, UFABC, PUCPR, UFMS, UnB, PUC-RS, PUCMG, UNIRIO, UFS e UFU.

Contato
Universidade Federal de Mato Grosso do Sul (UFMS)
Faculdade de Computação
Caixa Postal 549
CEP 79.070-900 Campo Grande - MS - Brasil
http://www.larc.org.br

Organização do SBRC 2019

Coordenadores Gerais
Alberto Egon Schaeffer Filho (UFRGS)
Weverton Luis da Costa Cordeiro (UFRGS)

Coordenadores do Comitê de Programa
Antônio Jorge Gomes Abelém (UFPA)
Fabíola Gonçalves Pereira Greve (UFBA)

Coordenador de Palestras e Tutoriais
Ítalo Cunha (UFMG)

Coordenador de Painéis e Debates
Artur Ziviani (LNCC)

Coordenador de Minicursos
Miguel Elias Mitre Campista (UFRJ)

Coordenador de Workshops
Stênio Fernandes (UFPE)

Coordenador do Salão de Ferramentas
Leandro Villas (UNICAMP)

Coordenador do Concurso de Teses e Dissertações
Daniel Fernandes Macedo (UFMG)

Coordenadores do Hackaton
Raquel Lopes (UFMG)
Luis Carlos de Bona (UFPR)

Comitê de Organização Local
Avelino Zorzo (PUCRS)
Carlos Raniery Paula dos Santos (UFSM)
Cristiano Bonato Both (Unisinos)
Guilherme Rodrigues (IFSUL Charqueadas)
Jéferson Campos Nobre (UFRGS)
Juliano Wickboldt (UFRGS)
Marcelo Caggiani Luizelli (Unipampa)
Marcelo da Silva Conterato (PUCRS)
Rafael Pereira Esteves (IFRS)
Rodrigo Mansilha (Unipampa)
Tiago Ferreto (PUCRS)
Vinícius Guimarães (IFSUL Charquadas)

Comite Consultivo
Fabíola Gonçalves Pereira Greve (UFBA)
Paulo André da Silva Gonçalves (UFPE)
Fábio Luciano Verdi (UFSCAR)
Jó Ueyama (USP)
Antônio Jorge Gomes Abelém (UFPA)
Eduardo Cerqueira (UFPA)
Luiz Fernando Bittencourt (Unicamp)
Rossana Andrade (UFC)
Michele Nogueira (UFPR)
Edmundo Madeira (UNICAMP)

Mensagem do Coordenador de Minicursos

É com muita satisfação que apresento os Minicursos do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), realizado entre os dias 6 e 10 de maio de 2019, na linda cidade de Gramado - RS. Os Minicursos do SBRC produzem os capítulos do Livro de Minicursos que tradicionalmente trazem material de alta qualidade, abordando temas de pesquisa e desenvolvimento de interesse da comunidade. O livro, assim como as apresentações no simpósio, tem sempre um viés altamente didático, capaz de motivar alunos de graduação, pós-graduação e profissionais da área a mergulharem nos temas propostos. A ideia é abrir horizontes e complementar a formação dos participantes em temas recentes que possivelmente ainda não são cobertos nas grades curriculares das instituições tradicionais de ensino e pesquisa.

Neste ano, 18 registros/submissões de propostas foram recebidos, levando em conta o prazo final, que ocorreu após dois adiamentos. Os adiamentos foram realizados para que mais submissões pudessem surgir e, de fato, surgiram. Todas as propostas foram atribuídas a, no mínimo, três revisores que aceitaram e revisaram no prazo, mesmo em época de férias. Gostaria de agradecer imensamente aos 24 membros do comitê deste ano que se dedicaram à revisão das propostas de forma sempre atenciosa. Foi um prazer trabalhar com todos.

Dos 18 registros/submissões de propostas, decidiu-se, por questões de espaço no evento, aceitar no máximo seis propostas. Esse número atende o critério de 33% adotado inclusive na trilha principal do SBRC e, portanto, foi julgado razoável. As seis propostas selecionadas, aquelas que tiveram as maiores médias gerais, são certamente ótimas candidatas a atrair público ao evento, já que abordam temas como blockchains, virtualização de redes, computação urbana, processamento de pacotes, redes sem fio, criptografia e ciência de dados. Vale mencionar que, entre as propostas não contempladas, algumas teriam plenas condições de serem aceitas, o que infelizmente não foi possível desta vez.

Gostaria de agradecer aos organizadores gerais do SBRC, Professores Alberto Egon Schaeffer Filho e Weverton Cordeiro, que com muita dedicação e empenho aceitaram o desafio de organizar o SBRC de 2019. Agradeço-os especialmente pela confiança na organização dos Minicursos, no trato sempre muito gentil e na infinita paciência. Agradeço ainda ao comitê local, em especial ao Jonatas Marques e aos Professores Carlos Raniery e Vinícius Guimarães por toda a ajuda. Todos os e-mails enviados (e não foram poucos) foram sempre respondidos prontamente.

Agradeço também aos autores dos minicursos, pois sem a dedicação e esmero de todos na escrita e na entrega pontual do material, nada disso seria viável. Tenho certeza que todas as apresentações brindaram todo o trabalho e possibilitaram plena absorção técnica da plateia. O desfecho dos minicursos foi suave graças à colaboração e ao profissionalismo de todos. Agora, é rentabilizar o árduo trabalho fomentando frutíferas discussões em direção ao avanço da ciência nacional.

Miguel Elias Mitre Campista
Coordenador de Minicursos do SBRC 2019

Comitê de Avaliação

- Alex Vieira (UFJF)
- Alfredo Goldman (USP)
- Anelise Munaretto (UTFPR)
- Artur Ziviani (LNCC)
- Christian Esteve Rothenberg (UNICAMP)
- Daniel Batista (USP)
- Daniel Fernandes Macedo (UFMG)
- Daniel Guidoni (UFSJ)
- Daniel Menasche (UFRJ)
- Danielo G. Gomes (UFC)
- Edmundo Madeira (UNICAMP)
- Eduardo Cerqueira (UFPA)
- Fabíola Greve (UFBA)
- Gustavo Figueiredo (UFBA)
- Igor Moraes (UFF)
- José Augusto Suruagy Monteiro (UFPE)
- Luciano Paschoal Gaspary (UFRGS)
- Marcelo Dias de Amorim (Sorbonne Université – CNRS)
- Marcelo Rubinstein (UERJ)
- Marinho Barcellos (UFRGS)
- Michele Nogueira (UFPR)
- Miguel Elias M. Campista (UFRJ)
- Rodrigo de Souza Couto (UFRJ)
- Ronaldo Ferreira (UFMS)

Sumário

Capítulo 1

Segurança na Internet do Futuro: Provendo Confiança Distribuída através de Correntes de Blocos na Virtualização de Funções de Rede

Gabriel Antonio F. Rebello, Gustavo F. Camilo, Leonardo G. C. Silva, Lucas A. C. de Souza, Lucas C. B. Guimarães e Otto Carlos M. B. Duarte

Grupo de Teleinformática e Automação/PEE-COPPE/DEL-Poli
Universidade Federal do Rio de Janeiro - RJ, Brasil

Resumo

A Internet do Futuro, além de interconectar bilhões de dispositivos, deverá atender com agilidade a uma enorme diversidade de requisitos de serviços, alavancar novas aplicações e continuar sendo a maior infraestrutura para crescimento econômico e social das nações. Esta Internet de nova geração, baseada na tecnologia de Virtualização de Funções de Rede (Network Function Virtualization - NFV) e Redes Definidas por Software (Software-Defined Networking - SDN) será a essência das redes móveis de quinta geração (5G) e das cidades inteligentes (smart cities). Um dos desafios fundamentais de pesquisa é garantir o provisionamento seguro de serviços em um ambiente de nuvem onde múltiplos inquilinos e usuários sem confiança mútua compartilham recursos. O objetivo deste capítulo é mostrar de forma clara, objetiva e sucinta os fundamentos-chave de corrente de blocos (blockchain), e relacionar estes conceitos aos desafios de pesquisa em segurança nas redes de nova geração. Além disso, este capítulo dedica uma parte significativa à realização de uma atividade prática que mostra o desenvolvimento de duas correntes de blocos baseadas na plataforma Hyperledger Fabric. As correntes de blocos mitigam ataques em uma arquitetura de fatiamento de redes que provê o encadeamento de funções de rede para construir serviços fim-a-fim sob demanda. Os participantes poderão criar, verificar e discutir o funcionamento de cada componente de uma corrente de blocos baseada no Hyperledger Fabric e aprender a controlar as atividades através de contratos inteligentes.

Este capítulo foi realizado com recursos do CNPq, CAPES, FAPERJ e FAPESP (2015/24514-9, 2015/24485-9 e 2014/50937-1).

1.1. Introdução

A Internet do Futuro interconectará bilhões de dispositivos através de sistemas construídos a partir de milhares de serviços distribuídos. Aglomerados com um enorme número de máquinas físicas e virtuais manipularão grandes quantidades de dados gerados a partir de inúmeros sensores e atuadores da Internet das Coisas (*Internet of Things* - IoT), gerando um volume de informações jamais visto. A Internet será constituída de fatias de rede (*network slices*) adaptadas a cada tipo de aplicação, que serão criadas, alteradas e destruídas de maneira ágil e escalável para oferecer alta disponibilidade com baixo custo de operação. Esta Internet de nova geração, baseada na tecnologia de Virtualização de Funções de Rede (*Network Function Virtualization* - NFV) e Redes Definidas por Software (*Software-Defined Networking* - SDN), será a essência das redes móveis de quinta geração (5G) e das cidades inteligentes (*smart cities*).

Neste novo paradigma, um dos desafios fundamentais de pesquisa é garantir o provisionamento seguro de serviços em um ambiente de nuvem onde múltiplos inquilinos e usuários sem confiança mútua compartilham recursos. Uma forma de garantir a confiança de forma distribuída e permitir a análise forense para identificação dos responsáveis pelos problemas é através do uso da tecnologia de corrente de blocos (*blockchain*). A corrente de blocos, conhecida por suas aplicações em criptomoedas como o Bitcoin [Nakamoto 2008] e Ethereum [Wood 2014], é uma tecnologia disruptiva que deve revolucionar não somente a tecnologia da informação como também a economia e a sociedade, permitindo uma maior descentralização do mundo atual. Pode-se prever um futuro no qual a computação e a Internet associadas à tecnologia de corrente de blocos permitirão uma computação planetária distribuída que executa aplicações e contratos inteligentes através de um consenso entre computadores sem nenhuma entidade intermediária. Esta confiança distribuída sem intermediários provida pela corrente de blocos permitirá múltiplas ofertas de funções virtualizadas de rede e seleção criteriosa entre diversos provedores de infraestrutura, produzindo uma concorrência sem precedentes na área de telecomunicações.

As tecnologias de corrente de blocos de primeira geração, baseadas no Bitcoin [Nakamoto 2008], e de segunda geração, baseadas nos contratos inteligentes do Ethereum [Wood 2014] já revolucionaram o mundo atual ao criar uma camada de confiança para transferências seguras de ativos e execução segura de contratos. No entanto, para a Internet do Futuro os desafios são ainda maiores, pois prevê-se que em 2025 o uso de dispositivos IoT gere até 3 trilhões de dólares em valor de mercado [Arnott et al. 2016] e atinja a marca de 75 bilhões de dispositivos conectados [Statista 2018], que atenderão a diversas aplicações, desde redes veiculares tolerantes a atraso até serviços críticos como saúde eletrônica (*e-Health*), cidades inteligentes (*smart cities*) e redes elétricas inteligentes (*smart grids*).

Este capítulo aborda as tecnologias de virtualização de funções de rede, de redes definidas por software e de corrente de blocos para prover segurança à Internet do Futuro em um ambiente multiusuário e multi-inquilino. O minicurso foca na criação e gerenciamento distribuído de fatias de redes, de forma ágil, confiável e segura, através do encadeamento de funções de rede em um cenário de múltiplos centros de dados com diversos provedores de funções virtuais de rede. O minicurso é organizado em quatro partes principais: i) a tecnologia de virtualização de funções de redes; ii) a tecnologia de

corrente de blocos; iii) fatiamento seguro de redes através de corrente de blocos e iv) a elaboração prática de uma aplicação que se serve de uma corrente de blocos programada no Hyperledger Fabric [2]. A aplicação desenvolvida na parte prática segue as funcionalidades providas pelo sistema SINFONIA (*Secure vIrtual Network Function Orchestrator for Non-repudiation, Integrity, and Auditability*) [Rebello et al. 2018], um sistema desenvolvido pelo Grupo de Teleinformática e Automação (GTA) que usa corrente de blocos para garantir segurança na criação das redes. O protótipo desenvolvido também oferece funcionalidades de garantia de análise forense de configuração, gerenciamento e migração de máquinas virtuais [Alvarenga 2018].

O objetivo deste capítulo é mostrar de forma clara, objetiva e sucinta os fundamentos chave de corrente de blocos, e relacionar estes conceitos aos desafios de pesquisa em segurança nas redes de nova geração. O principal diferencial deste minicurso é a utilização da corrente de blocos em ambientes de rede virtualizados, analisando os impactos, provendo segurança e gerando oportunidades de pesquisa para o futuro.

1.2. Virtualização de Redes e Segurança

Esta seção apresenta os conceitos básicos da virtualização de funções de rede (*Network Function Virtualization* - NFV) e do encadeamento de funções de serviço (*Service Function Chaining* - SFC), discutindo os principais desafios de segurança em ambientes multi-inquilino e multiusuário característicos da virtualização.

1.2.1. Virtualização de Funções de Rede

A tecnologia de virtualização de funções de rede (*Network Function Virtualization* – NFV) é a aplicação de conceitos de virtualização e computação em nuvem para as telecomunicações [Medhat et al. 2017]. O NFV permite reduzir gastos e flexibilizar o gerenciamento e a operação das redes de comunicações através da substituição de recursos em *hardware* dedicado por funções virtualizadas em *software*, que são executadas em *hardware* de uso geral [Pattaranantakul et al. 2016]. Assim, os sistemas intermediários (*middleboxes*), como *firewalls*, sistemas de detecção e prevenção de intrusão, balanceadores de carga, roteadores, *proxies*, etc. que hoje são implementados em *hardware* específicos de um fabricante, serão substituídos por funções em *software* virtualizadas que podem ser providas por diferentes fornecedores [Sekar et al. 2012]. Dentre os principais benefícios da tecnologia NFV estão a redução dos gastos de capital (*CAPital EXpenditure* - CAPEX) e gastos operacionais (*OPerational EXpenditure* - OPEX) com equipamentos dedicados que requerem alocação de espaço físico, refrigeração adequada, gasto energético e formação de recursos humanos. Outra importante vantagem da tecnologia NFV é o menor tempo de chegada até o mercado (*time-to-market* - TTM) para uma função de rede desde a sua concepção e desenvolvimento até a implementação no domínio de um operador de rede [Mijumbi et al. 2016]. Estima-se reduzir o TTM, que hoje usualmente leva quatro ou cinco anos, para três a quatro meses.

Algumas definições são reforçadas para o melhor entendimento desta proposta. O orquestrador de nuvem é a entidade que cria a cadeia de funções de rede através da plataforma de virtualização utilizada no centro de dados. O administrador da nuvem é a

[2] Disponível em https://github.com/hyperledger/fabric

companhia ou a operadora responsável por um ou mais centros de dados, na qual a virtualização é realizada e que cada orquestrador implementa suas funções. Um inquilino é um cliente do centro de dados que usufrui de um serviço provido pelo encadeamento de funções de rede. É possível ainda orquestrar diferentes cadeias de funções que possuem uma mesma VNF em comum, ou seja, fatias de recursos de uma VNF podem ser compartilhadas entre inquilinos. Dessa forma, um ambiente de nuvem com infraestrutura de NFV é definido como um ambiente multi-inquilino, o que permite uma gama de novos ataques, como *side-channel*, inspeção de tráfego, esgotamento de recursos etc. Portanto, requisitos de segurança adicionais são necessários para prover o correto isolamento entre todos os inquilinos em um cenário NFV.

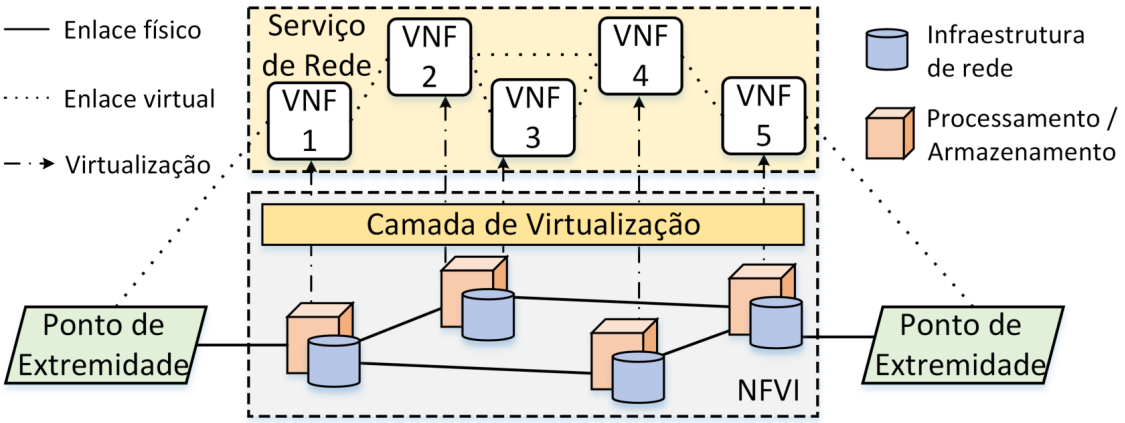


Figura 1.1. A infraestrutura de virtualização de funções de rede, que permite criar enlaces virtuais para serviços fim-a-fim através do encaminhamento de pacotes entre funções virtuais de rede.

A infraestrutura de virtualização de funções de rede (*NFV Infrastructure* – NFVI), proposta pelo *European Telecommunications Standards Institute* (ETSI) na principal arquitetura de gerência e orquestração das funções virtuais de rede (*Network Function Virtualization MANagement and Orchestration* – NFV-MANO) [ETSI 2014], tem como objetivo padronizar a composição de serviços fim-a-fim sob medida para cada aplicação. A NFVI fornece as abstrações de processamento, armazenamento e acesso à rede para que máquinas virtuais se comportem como funções virtuais de rede. A Figura 1.1 mostra uma simplificação de uma rede virtualizada através da infraestrutura de virtualização. Na arquitetura, o NFVI transforma os nós de processamento, os nós de armazenamento e a infraestrutura de rede de uma infraestrutura física em uma camada de virtualização que aloca os recursos necessários para cada função virtualizada. Na arquitetura do NFV-MANO, cada provedor de infraestrutura administra centros de dados com NFVIs capazes de realizar operações de gerenciamento e orquestração de funções virtualizadas de rede. As funções virtualizadas podem ser encadeadas através de enlaces virtuais para prover um serviço fim-a-fim entre duas extremidades da rede.

1.2.2. Encadeamento de Funções de Serviço

Nas redes da próxima geração, baseadas na tecnologia NFV [ETSI 2014], o encadeamento de funções de serviço (*Service Function Chaining* – SFC) [Halpern and Pignataro 2015] é o procedimento fundamental para fornecer controle e gerenciamento flexível

do tráfego de um serviço ou de uma aplicação. Uma função de serviço é uma função de rede virtualizada (*Virtual Network Function* - VNF) ou não virtualizada que compõe um serviço fim-a-fim. Por simplicidade, no cenário abordado deste trabalho, consideram-se funções de serviço como sinônimos de funções virtualizadas de rede. Alguns pesquisadores, no entanto, consideram uma função de serviço como uma composição de uma ou mais funções de rede encadeadas [Li and Chen 2015]. O encadeamento de funções de serviço no caminho da origem até o destino fornece, sob demanda, um serviço adaptado para cada aplicação ou usuário.

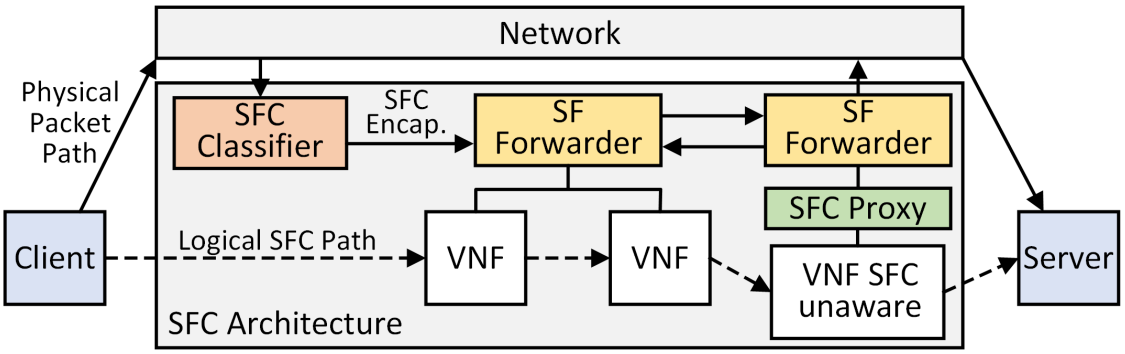


Figura 1.2. Arquitetura-padrão do encadeamento de funções de serviço [Halpern and Pignataro 2015]. O classificador SFC encapsula o tráfico ingressante e encaminhador de funções de serviço (SFF) encaminha o tráfego encapsulado para a cadeia correta baseado no cabeçalho SFC. Um proxy SFC realiza o encapsulamento e desencapsulamento para funções de rede sem suporte ao cabeçalho SFC.

A Figura 1.2 ilustra a arquitetura padrão do encadeamento de funções de serviço, proposta na RFC 7665 [Halpern and Pignataro 2015]. Considerando um ambiente multisserviço, configura-se um classificador com regras específicas para identificar e especificar a cadeia de VNFs que cada fluxo de serviço deve atravessar. A cadeia de funções de serviço é definida por uma sequência lógica e ordenada de VNFs, chamado caminho de função de serviço (*Service Function Path* - SFP) e cada cadeia possui um identificador. Assim, fluxos de diferentes serviços podem ser isolados e atravessar simultaneamente uma mesma VNF. As funções virtualizadas de rede também podem ser hospedadas em diferentes nós. Portanto, deve existir um encaminhador de função de serviço (*Service Function Forwarder* - SFF) em cada nó da infraestrutura de virtualização de funções de rede (NFVI) para fornecer enlaces virtuais para as VNFs hospedadas. Na implementação padrão de encadeamento de funções de serviço, o isolamento entre fluxos de serviços que atravessam a mesma VNF é realizado através de encapsulamento de pacotes. VNFs podem estar cientes ou inconscientes do encapsulamento SFC. VNFs que desconhecem o encapsulamento requerem um elemento precedente, o Proxy SFC, para desencapsular pacotes SFC e transformá-los em pacotes comuns da pilha TCP/IP. VNFs conscientes do SFC realizam o encapsulamento e desencapsulamento através de um módulo do *kernel* ou um comutador virtualizado compatível com o encapsulamento SFC.

1.2.3. Segurança em Ambientes de Funções Virtuais de Rede

Funções de rede podem ser estrategicamente posicionadas de acordo com sua função. Uma VNF de um sistema de detecção de intrusão (VNF IDS) é mais eficaz quando instanciada próximo de uma fonte de ataques ou em um centro de dados especializado em monitoramento de tráfego que realiza a correlação de informações e registros de detecção com outras VNF IDS. Uma VNF que realiza transcodificação de serviços de fluxo (*streaming*) deve estar localizada próximo à fonte do conteúdo para evitar perdas de dados na rede. Ainda, os requisitos de posicionamento de VNFs tornam-se pontos críticos quando não há um centro de dados da operadora na localização desejada, necessitando o uso de recursos virtuais de outra operadora mais próxima. Portanto, neste ambiente de nuvem flexível, uma cadeia de funções de rede pode ter componentes instanciados em domínios de operadoras diferentes. Em um cenário virtualizado, a comunicação entre pontos de extremidade é realizada através da conexão entre grandes centros de dados (*data centers*), que são capazes de prover, sob demanda, serviços fim-a-fim adaptados a cada aplicação através do encadeamento de funções de serviço (*Service Function Chaining* - SFC). Um cenário de encadeamento de funções para um serviço de rede é ilustrado na Figura 1.3. Neste cenário, uma cadeia de funções de rede é composta por VNFs hospedadas em diferentes domínios de operadoras de telecomunicações. Cada operadora possui um centro de dados, que contém um único orquestrador de VNFs e uma infraestrutura de virtualização baseada em máquinas de propósito geral que pode hospedar máquinas virtuais e VNFs.

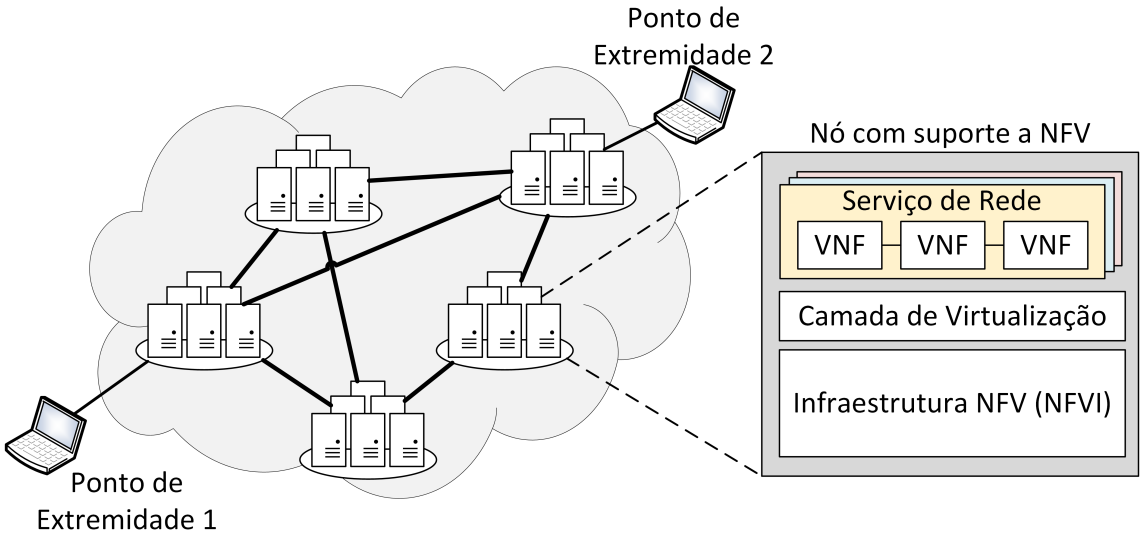


Figura 1.3. O cenário da virtualização de funções de rede. Uma conexão fim-a-fim entre dois pontos de extremidade atravessa conexões entre centros de dados que possuem infraestruturas de virtualização. Cada centro de dados possui uma infraestrutura de virtualização que gerencia e encadeia funções virtualizadas de rede para prover diferentes serviços fim-a-fim.

A virtualização de funções de rede e o encadeamento de funções de serviço provê a flexibilidade, a agilidade e o baixo custo desejado para as telecomunicações, mas traz novos desafios de segurança [Medhat et al. 2017]. Neste novo cenário, os inquilinos compartilham a mesma infraestrutura de nuvem, e cadeias podem envolver funções virtu-

alizadas instanciadas em domínios de operadoras concorrentes [Han et al. 2015, Mijumbi et al. 2016]. Desta forma, é necessário garantir que a cadeia de funções virtualizadas de rede seja construída de maneira confiável em um ambiente sem confiança entre os pares, sejam estes inquilinos ou domínios. O ambiente multi-inquilino e multi-domínio aumenta a possibilidade de ataques dentro da nuvem e dificulta a responsabilização ou uma possível indenização pelos provedores do serviço devido a um mau funcionamento. Além disso, os impactos de possíveis ataques tornam-se bem maiores, uma vez que ataques ao hospedeiro de funções de rede podem comprometer milhares de usuários simultaneamente [Bouras et al. 2017].

1.2.3.1. O modelo FCAPS e Terminologia

O modelo FCAPS (*Fault, Configuration, Administration, Performance and Security)* é o padrão de gestão de redes de telecomunicação [Raman 1998] em ambientes centralizados e de provedor único. O modelo provê padrões e boas práticas para assegurar propriedades fundamentais de segurança, como autenticidade, integridade e não-repúdio. No entanto, o cenário de virtualização de funções de rede é um ambiente distribuído que não possui confiança entre os pares, de forma que o FCAPS não é diretamente aplicável. Neste cenário, diversos provedores oferecem serviços distribuídos através de múltiplas nuvens, trazendo novos desafios, tais como: i) selecionar as métricas que garantem a corretude do serviço fim-a-fim de uma cadeia de funções de rede; ii) identificar o provedor de nuvem, dentre os participantes de uma cadeia de funções de rede, responsável por uma falha; e iii) estabelecer a proveniência, o impacto e o tempo que uma determinada falha permaneceu indetectada. Portanto, uma solução com o uso de corrente de blocos é apropriada para solucionar estes desafios.

1.2.3.2. Modelo de Atacante

Este trabalho considera um modelo de atacante de Dolev [Dolev and Yao 1983], no qual um atacante pode *ler*, *enviar* e *descartar* uma transação endereçada à corrente de blocos, ou qualquer pacote de rede. O atacante pode agir de maneira passiva, se conectando na rede e capturando toda troca de mensagens, ou de maneira ativa, injetando, repetindo, filtrando ou trocando informações. Os ataques podem atingir inquilinos, VNFs, a corrente de blocos ou a rede.

Ataques à corrente de blocos consistem em impedir que uma transação ou bloco legítimo seja incorporado à corrente de blocos. Para realizar este tipo de ataque, o atacante necessita controlar uma parcela significativa da rede para afetar o consenso. Esse tipo de ataque é mitigado pela tolerância a falhas do protocolo de consenso utilizado.

Ataques ao encadeamento de funções de rede consistem em modificar de forma maliciosa uma cadeia de funções de serviço. Este tipo de ataque é realizado por provedores que gerenciam as cadeias de funções de serviço e as funções virtualizadas de rede. Dentre os possíveis ataques estão o desvio de tráfego de uma cadeia de funções através de uma VNF maliciosa, a remoção ou modificação de uma VNF em uma cadeia, ou a sub-alocação de recursos para VNFs e enlaces virtuais.

Ataques a inquilinos ou VNFs consistem em personificar o alvo emitindo ou retransmitindo transações. Ataques de personificação são mitigados através do uso de chaves criptográficas assimétricas para assinar toda transação enviada à corrente de blocos. Toda transação é assinada por seu emissor e verificada pelos participantes do consenso. Este trabalho não trata casos de comprometimento de um inquilino ou VNF através da invasão de seu terminal ou roubo de suas chaves privadas. No entanto, a arquitetura proposta neste trabalho permite a auditoria das transações caso o atacante realize um ataque de personificação utilizando pares de chaves roubados. Outro trabalho relacionado propõe mitigar ataques a VNFs eliminando qualquer serviço de escuta ativo em uma VNF e utilizando seus terminais apenas em modo de leitura [Alvarenga et al. 2018]. Pares de chaves roubados ou perdidos podem ser substituídos para impedir maiores danos.

Ataques à rede consistem em isolar um ou mais inquilinos ou VNFs da rede, impedindo a emissão de transações e leitura do conteúdo da corrente de blocos. Essa categoria de ataque contempla ataques clássicos de rede, que podem ser mitigados através do estabelecimento de caminhos redundantes entre a corrente de blocos e as VNFs ou inquilinos. O foco deste trabalho não são esses ataques, e sim os ataques à corrente de blocos, aos inquilinos, às VNFs, e ao encadeamento de funções de rede.

Este capítulo foca no uso da tecnologia de corrente de blocos para mitigar possíveis vetores de ataques ao encadeamento de funções de rede, ataques aos inquilinos, ataques às VNF e ataques à própria corrente de blocos.

1.3. A Tecnologia de Corrente de Blocos[3]

Esta seção aborda a tecnologia de corrente de blocos, iniciando com o problema do gasto duplo resolvido em 2008 por Satoshi Nakamoto com a proposta da moeda virtual Bitcoin [Nakamoto 2008]. Aborda-se também propriedades e categorias de correntes de bloco, modelos de rede, assim como tipos de consenso e classes de protocolos de consenso.

1.3.1. O Problema do Gasto Duplo

Durante a maior parte do século XX, engenheiros, desenvolvedores, criptógrafos e profissionais de tecnologia da informação procuraram resolver o problema de transferência de ativos para permitir a criação de uma moeda digital descentralizada. A transferência de arquivos na Internet é realizada facilmente copiando-se o arquivo original e enviando-o para o destino. No entanto, a transferência de ativos (dinheiro, ações etc.) é bem mais complexa e requer um intermediário para prover confiança porque ao transferir um ativo da origem para o destino deve-se subtrair o valor do ativo a ser transferido da origem e acrescentá-lo no destino. A dificuldade reside no problema do gasto duplo (*double spending problem*), que ocorre quando um mesmo ativo é utilizado mais de uma vez em diferentes transações de um sistema, pois os ativos digitais podem ser facilmente replicados. Ainda que comumente postulado para o caso de moedas digitais, o problema estende-se a qualquer tipo de recurso digital único, como endereços IP, domínios, registros de identidade, propriedade intelectual, recursos computacionais, serviços, etc. A Figura 1.4 ilustra o problema do gasto duplo. No exemplo, suponha que Alice deseja transferir moedas para

[3]Esta seção é baseada no projeto de fim de curso de Gabriel Antonio Fontes Rebello [Rebello 2019].

Bob. Se Alice e Bob utilizam dinheiro em espécie, Alice deve ceder suas moedas a Bob e não possuirá mais as moedas após a transação. Porém, se uma moeda digital for utilizada, é necessária uma maneira de garantir que Alice gastou as moedas, pois moedas digitais são representações em bits que podem ser facilmente duplicados. Neste caso, Alice pode enviar um arquivo digital com o valor desejado a Bob enquanto mantém uma cópia local do arquivo. Se Alice ainda mantiver uma cópia, ela pode enviá-la a uma terceira pessoa, Carol, realizando um gasto duplo.

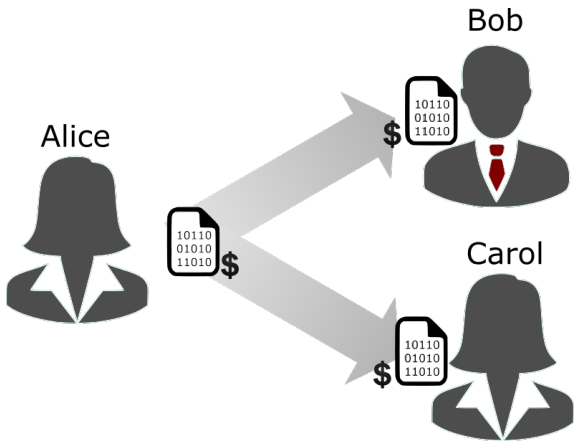


Figura 1.4. Exemplo do problema do gasto duplo no qual a mesma representação digital de uma moeda é replicada e gasta duas vezes.

Tradicionalmente, a prevenção do gasto duplo é realizada através da intermediação centralizada em uma terceira entidade com poderes de autoridade, que utiliza regras de negócio para autorizar as transações e verificar se as moedas envolvidas foram gastas. Essa abordagem é normalmente utilizada em bancos, companhias de crédito, plataformas de vendas em linha (*online*) e, no caso de ativos na Internet, através de organizações como a *Internet Assigned Numbers Authority* (IANA)[4]. Porém, a centralização implica que todos os participantes do sistema possuam total confiança na autoridade central, que pode cobrar taxas, limitar o volume de transações e controlar a rede de forma arbitrária. Além disso, uma falha na autoridade central pode interromper todas as transações do sistema por tempo indeterminado. O modelo centralizado, portanto, cria um ponto único de falha na rede, impactando tanto a disponibilidade quanto a confiança no sistema. No caso de ativos financeiros, os bancos cobram taxas exorbitantes e introduzem enormes atrasos para a transferência.

1.3.2. Criptomoedas e Corrente de Blocos

O conceito de moeda digital descentralizada, doravante referida como criptomoeda, é o principal propulsor de aplicações envolvendo trocas de ativos. Durante décadas, procurou-se um mecanismo para viabilizar uma criptomoeda totalmente funcional. Em 1983, David Chaumian introduziu o primeiro protocolo anônimo para criação de criptomoedas utilizando o conceito de assinatura cega (*blind signature*) [Chaum 1983]. A assinatura cega provê privacidade às transferências de moedas, mas depende fortemente de entidades centralizadas para realizar as assinaturas. Em 1998, Wei Dai propôs *b-money*,

[4]Disponível em https://www.iana.org/

uma criptomoeda que introduz a ideia de criar valor monetário através da resolução de desafios computacionais e com consenso descentralizado [Dai 1998]. No entanto, a proposta possuía poucos detalhes sobre a implementação do consenso, dificultando sua adoção. Em 2002, Adam Back propôs formalmente *Hashcash*, um algoritmo de prova de trabalho (*Proof of Work* - PoW) baseado em funções resumo (*hash*) criptográficas[5] para prevenção de *spam* de e-mails e ataques de negação de serviço [Back 2002]. Em 2005, Hal Finney desenvolveu o conceito de provas de trabalho reutilizáveis (*Reusable Proof of Work* - RPoW) em um sistema que reúne os fundamentos do *b-money* e os desafios computacionalmente custosos do *Hashcash* [Finney 2005]. No entanto, a proposta também dependia de entidades confiáveis para ser implementada. Finalmente, em 2008, Satoshi Nakamoto publicou "Bitcoin: A peer-to-peer electronic cash system," propondo e implementando pela primeira vez uma criptomoeda totalmente descentralizada, combinando a prova de trabalho com uma nova e revolucionária estrutura de dados organizada em blocos de transações: a corrente de blocos (*blockchain*) [Nakamoto 2008]. O mais impressionante é que a proposta de solução do problema utiliza tecnologias já conhecidas e dominadas como criptografia, consenso, redes de comunicação e incentivos.

1.3.2.1. Corrente de Blocos

A corrente de blocos, conhecida por suas aplicações em criptomoedas como o Bitcoin [Nakamoto 2008], Ethereum [Wood 2014] e outras tantas, é uma tecnologia disruptiva que deve revolucionar não somente a tecnologia da informação como também a economia e a sociedade, permitindo uma maior descentralização do mundo atual. O principal diferencial de uma corrente de blocos é ser uma estrutura de dados distribuída que garante confiabilidade sem necessitar de uma autoridade central comum a todas as entidades. Pela primeira vez, duas partes que não confiam uma na outra ou mesmo que não se conhecem podem trocar ativos sem um intermediário. A corrente de blocos substitui o modelo centralizado por um modelo colaborativo no qual a maior parte da rede deve concordar sobre todas as decisões. A corrente de blocos ao permitir decisões coletivas, eliminando a entidade centralizadora, potencialmente torna desnecessário governos, bancos, agências de crédito etc. para transferência de ativos, fornecendo poder à população. A corrente de blocos é aplicável em todo ambiente distribuído no qual os participantes não possuem confiança mútua e também não confiam ou são incapazes de entrar em acordo sobre uma autoridade centralizada que governe todos os procedimentos sensíveis da rede [Wüst and Gervais 2018]. Suas propriedades, discutidas com maior profundidade na Seção 1.3.2.3, garantem a auditabilidade, a imutabilidade e o não repúdio de todas as transações realizadas por participantes da rede, e são chave para a criação de um ambiente seguro e escalável.

A corrente de blocos é uma tecnologia de livro-razão distribuído (*Distributed Ledger Technology* - DLT) que utiliza máquinas independentes, denominadas de nós mineradores[6], para gravar, compartilhar e sincronizar transações em um sistema de controle

[5]Os fundamentos de funções resumo criptográficas e criptografia de chaves assimétricas são apresentados no Apêndice A.

[6]Os mineradores usualmente recebem uma recompensa para executarem a mineração e, em modelos de consenso sem recompensa, são chamados de participantes do consenso ou simplesmente de nós.

descentralizado sobre uma rede par a par (*peer-to-peer* - P2P). Cada nó minerador possui uma cópia local da corrente de blocos na qual pode verificar qualquer transação emitida na rede desde sua criação. A adição de blocos é realizada de maneira descentralizada através de um protocolo de consenso comum aos nós mineradores. Assim, um atacante, ou grupo de atacantes em conluio, que deseja realizar um gasto duplo precisa controlar uma parcela grande o suficiente da rede para propor, utilizando o protocolo de consenso, um bloco que oculte uma de suas transações [Eyal and Sirer 2018, Nakamoto 2008]. O protocolo de consenso garante que as cópias das correntes de blocos são a mesma em qualquer nó minerador e, portanto, a propriedade de não repúdio de transações é garantida. Todas as transações são assinadas por seus emissores através de criptografia de chaves assimétricas e, na maior parte das implementações, utiliza-se a chave pública como identificador do nó na rede.

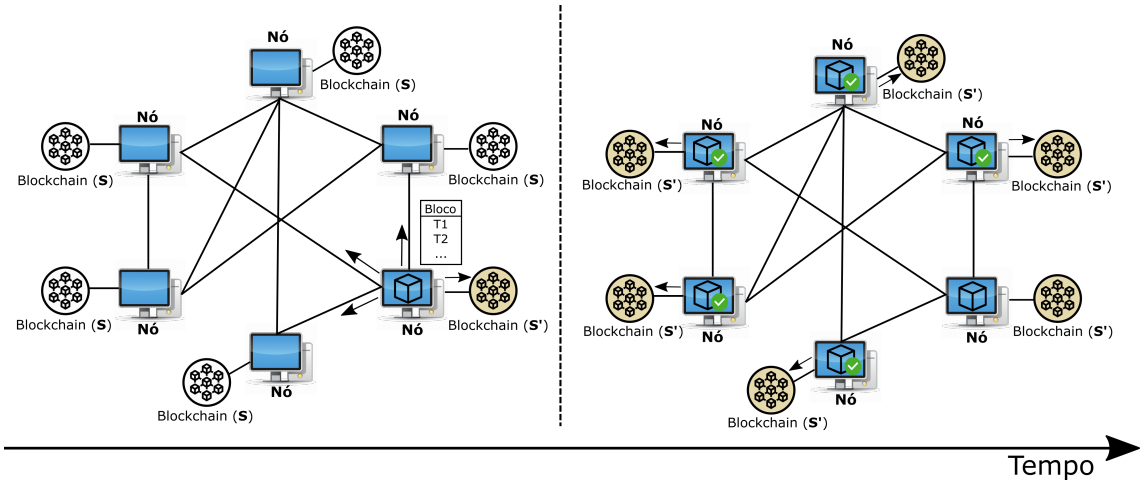


Figura 1.5. Atualização de uma corrente de blocos (*blockchain*) através de um protocolo de consenso genérico. Um nó minerador propõe um novo bloco em difusão e os demais nós mineradores verificam e adicionam independentemente o bloco proposto à corrente de blocos, incrementando o estado global S de maneira consistente.

A Figura 1.5 ilustra o funcionamento de uma corrente de blocos como um livro-razão distribuído através de um protocolo de consenso genérico. Um nó minerador da rede que deseja inserir um novo bloco, alterando o estado atual S da corrente de bloco, inicia uma rodada de consenso reunindo as transações válidas recebidas de usuários em um novo bloco a ser proposto. A seguir, o bloco proposto é validado localmente de acordo com políticas pré-definidas e é difundido na rede. Dependendo do protocolo de consenso adotado, o bloco proposto pode ser adicionado imediatamente, de forma assíncrona, à corrente de blocos do nó minerador que propôs o bloco, ou de forma síncrona em todos os nós mineradores ao fim da rodada. A figura mostra o caso de um protocolo no qual o estado S é alterado para S' no nó minerador que propôs o bloco antes da difusão. Ao receber o bloco proposto, cada nó minerador valida o bloco independentemente e, caso aprove o bloco, o adiciona à corrente de blocos e chega ao estado S'. O algoritmo de validação de um bloco e de cada transação deve ser acordado previamente por todos os nós mineradores. Todo protocolo de consenso possui um mecanismo de atualização para obter o estado mais recente e corrigir discrepâncias de estado resultantes de blocos não

aprovados, garantindo a consistência da corrente de blocos em toda a rede. Os protocolos de consenso e suas premissas são melhor discutidos na Seção 1.3.3.

A Figura 1.6 mostra o ciclo de vida de uma transação em um sistema de troca de ativos digitais através de uma transação simplificada de Bitcoin (BTC) [Nakamoto 2008]. Para enviar ou receber ativos, um usuário deve utilizar um par de chaves assimétricas para assinar transações. Por exemplo, para transferir 0,5 BTC a Bob, Alice cria uma transação contendo: i) uma entrada com o seu endereço e o total de BTC que possui; ii) uma saída com o endereço de Bob e o valor que deseja transferir e iii) uma saída com seu próprio endereço e o valor de irá possuir após a transferência do valor desejado. A seguir, Alice utiliza um algoritmo criptográfico para assinar a transação, criando uma transação assinada, e a envia com sua chave pública em difusão para a rede par a par (*peer-to-peer* - P2P) na qual estão os nós mineradores. A partir deste momento, qualquer participante do consenso pode aplicar o algoritmo de validação da rede para verificar criptograficamente a autenticidade da transação e se a transação conflita com outra já registrada na corrente de blocos. Transações inválidas são descartadas imediatamente. Caso a validação ocorra com sucesso, o participante do consenso adiciona a transação a um novo bloco que é proposto na próxima rodada do protocolo de consenso. A maior parte dos sistemas de troca de ativos provê programas que gerenciam chaves, armazenam endereços e emitem transações assinadas de forma transparente ao usuário. Estes programas são amplamente conhecidos como "carteiras".

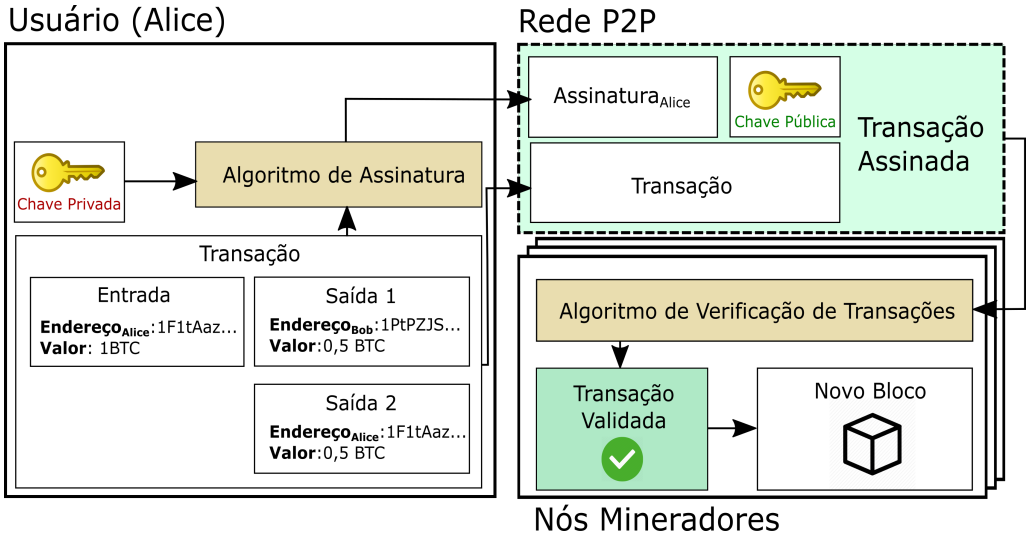


Figura 1.6. Ciclo de vida de uma transação em uma corrente de blocos. O usuário A difunde na rede par a par uma transação assinada que pode ser verificada e adicionada a um bloco por qualquer nó minerador.

1.3.2.2. Estruturas de Dados

A estrutura de dados da corrente de blocos proposta por Nakamoto como parte da criptomoeda Bitcoin [Nakamoto 2008] é uma lista encadeada de estruturas de dados menores, conhecidas como blocos. Cada bloco está conectado a seu antecessor através de uma função resumo (*hash*) criptográfica, criando uma corrente de blocos conforme

a mostra a Figura 1.7 A corrente de blocos funciona como um livro-razão (*ledger*), ou registro permanente (*log*), de blocos ordenados no tempo. Cada bloco na corrente possui um cabeçalho contendo o valor resultante de uma função resumo (*hash*) criptográfica do bloco antecessor, e uma seção de conteúdo que armazena dados relevantes a uma aplicação. A única exceção a esta regra é o bloco inicial, o gênesis, que por definição não possui bloco anterior. A utilização de uma sequência de funções resumo criptográficas, cujas saídas diferem drasticamente após a alteração de qualquer bit na entrada[7], implica que todos os blocos incluídos a partir de uma possível alteração devem ser reconstruídos. Assim, a adição de blocos torna cada vez mais custoso alterar a corrente e, como a corrente de blocos é replicada em cada nó minerador da rede, um atacante deve invadir todos os mineradores e recriar cada corrente de blocos para modificar uma transação passada. A replicação da corrente de blocos em todos os nós mineradores e o encadeamento através de funções *hash* criptográficas garante, portanto, a imutabilidade de um bloco com muitos sucessores.

As principais características da corrente de blocos são descritas em [Greve et al. 2018, Mello et al. 2017, Chicarino et al. 2017, Mattos et al. 2018]. A literatura de corrente de blocos apresenta propostas específicas que utilizam a seção de conteúdo do bloco para diferentes propósitos. Nakamoto propõe originalmente registrar transações bancárias para criar uma criptomoeda [Nakamoto 2008]. Wood *et al.*, Frantz *et al.* e Androulaki *et al.* propõem utilizar correntes de blocos para armazenar e executar contratos inteligentes através de uma máquina virtual distribuída [Wood 2014, Frantz and Nowostawski 2016, Androulaki et al. 2018]. Wilkinson *et al.* propõem um sistema baseado em correntes de blocos para prover armazenamento descentralizado em nuvem com privacidade [Storj Labs 2018, Wilkinson et al. 2014]. Ali *et al.* propõem o uso de correntes de blocos para registrar nomes de domínio [Ali et al. 2016]. Azaria *et al.* propõem registrar informações de fichas médicas [Azaria et al. 2016]. Lee *et al.* propõem registrar votos para criar um sistema de votação descentralizado [Lee et al. 2016]. Christidis e Hun *et al.* propõem rastrear dispositivos de Internet das Coisas (*Internet of Things* - IoT) através do armazenamento de informações na corrente de blocos [Christidis and Devetsikiotis 2016, Huh et al. 2017]. Bozic *et al.* e Alvarenga *et al.* propõem registrar informações de máquinas virtuais e funções virtualizadas de rede na corrente de blocos [Bozic et al. 2017, Alvarenga et al. 2018].

O principal conteúdo de uma corrente de blocos, no entanto, são as transações. Uma transação representa uma ação atômica a ser armazenada na corrente de blocos que transfere um ativo entre duas entidades. A transação possui quatro componentes principais: i) um remetente, que possui um ativo que deseja transferir e que emite a transação; ii) um destinatário que receberá o ativo que se deseja transferir; iii) o ativo que será transferido e iv) uma assinatura que corresponde a uma função *hash* de todos os campos anteriores pelo remetente. Quando a imutabilidade da corrente de blocos é utilizada com transações assinadas, cria-se um sistema que funciona como um livro-razão distribuído. As transações no sistema são ordenadas dentro de cada bloco e podem ser verificadas por qualquer nó participante da rede, desde o bloco inicial da corrente de blocos, denominado bloco gênesis. Pela propriedade de chaves criptográficas assimétricas, a assinatura do

[7]Os fundamentos de funções resumo criptográficas e criptografia de chaves assimétricas são apresentados no Apêndice A.

hash da transação, usando a chave privada do remetente provê autenticidade, não repúdio e integridade à transação, fornecendo maior segurança à rede. Ainda, é possível obter pseudo-anonimidade utilizando chaves públicas ou endereços únicos, que não revelam as identidades pessoais, para identificar remetentes e destinatários. A privacidade de transações, desejada em casos onde apenas o emissor e destinatário devem poder consultar a transação [Androulaki et al. 2018], pode ser garantida criptografando a transação com a chave pública do destinatário.

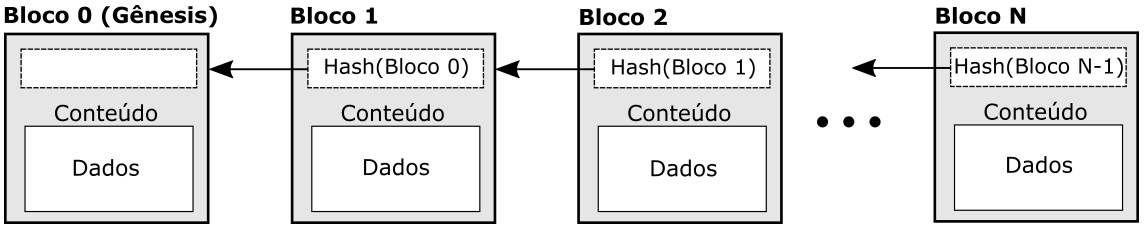


Figura 1.7. Estrutura de uma corrente de blocos na qual cada bloco é associado ao bloco seguinte e a integridade das transações de um bloco é garantida por uma uma função resumo (*hash*) criptográfica.

1.3.2.3. Propriedades de Corrente de Blocos

A corrente de blocos possui propriedades que proveem benefícios às aplicações e sistemas baseados nesta tecnologia. As principais propriedades da corrente de blocos são [Zheng et al. 2018, Xu et al. 2017, Wüst and Gervais 2018]:

- **Descentralização**. As correntes de blocos executam de maneira distribuída, através do estabelecimento de consenso entre todos os participantes da rede. Não há uma entidade centralizadora.
- **Desintermediação**. A corrente de blocos elimina a necessidade de um intermediário confiável para a troca de ativos. Os ativos podem ser trocados diretamente pela rede e a confiança é estabelecida através de consenso.
- **Imutabilidade**. Os dados armazenados em uma corrente de blocos são imutáveis. Não é possível modificar ou recriar qualquer dado incluído na corrente de blocos de forma retroativa. Toda atualização na corrente de blocos é realizada de forma incremental.
- **Irrefutabilidade**. Os dados são armazenados na corrente de blocos em forma de transações assinadas, que não podem ser alteradas devido à propriedade de imutabilidade da corrente de blocos. Portanto, o emissor de uma transação jamais pode negar sua existência.
- **Transparência.** Todos os dados armazenados na correntes de bloco são acessíveis por todos os participante da rede. Em correntes de blocos públicas, como o Bitcoin [Nakamoto 2008] e o Ethereum [Wood 2014], as transações são abertas para qualquer usuário com acesso à Internet.

- **Auditabilidade**. Como consequência da transparência, todo participante pode verificar, auditar e rastrear os dados inseridos na corrente de blocos para encontrar possíveis erros ou comportamentos maliciosos. No caso de correntes de blocos federadas, o processo de auditagem pode responsabilizar um malfeitor na rede.
- **Disponibilidade**. As correntes de blocos são estruturas replicadas em cada participante da rede e, portanto, a disponibilidade do sistema é garantida mesmo sob falhas, devido à redundância de informações.
- **Anonimidade**. Os usuários e nós mineradores de uma corrente de blocos são identificados por chaves públicas ou identificadores únicos que preservam suas identidades. Ainda, é possível utilizar uma chave pública em cada transação, evitando a rastreabilidade do usuário e conferindo um grau a mais de anonimidade.

Além das propriedades citadas, a corrente de blocos também pode prover privacidade, ao custo da transparência e auditabilidade. Em algumas implementações de corrente de blocos, os dados inseridos na corrente de blocos são criptografados com uma chave pública, evitando que todos os participantes possam ver o conteúdo da transação [Smolensk 2018, Androulaki et al. 2018]. Nesses casos, a transparência e a auditabilidade limitam-se a um ou mais participantes que detêm a chave privada correspondente e podem descriptografar a transação criptografada.

1.3.2.4. Categorias de Corrente de Blocos

A utilização de correntes de blocos constrói uma máquina de estados replicada que garante a consistência do sistema e um estado global compartilhado entre todos os participantes da rede. No entanto, a corrente de blocos deve ser configurada para atender às demandas de cada aplicação através de decisões de projeto. Através da taxonomia proposta na literatura [Christidis and Devetsikiotis 2016, Zheng et al. 2018, Xu et al. 2017] e das principais aplicações de correntes de blocos [Nakamoto 2008, Wood 2014, Androulaki et al. 2018], é possível categorizar as correntes de blocos em três tipos: **correntes de blocos públicas**, **correntes de blocos federadas** e **correntes de blocos privadas**. Os tipos de correntes de blocos diferenciam-se nos seguintes aspectos:

- **Permissão de escrita.** Correntes de blocos públicas, ou **não-permissionadas**, não impõem restrições de permissão a participantes da rede que desejam tornar-se mineradores e adicionar blocos através do protocolo de consenso [Nakamoto 2008, Wood 2014]. Em correntes de blocos federadas e privadas, também conhecidas como **permissionadas**, um novo nó minerador deve obter permissão de escrita dos demais mineradores através, respectivamente, de consenso ou de uma decisão centralizada.
- **Permissão de leitura.** Transações em uma corrente de blocos pública são publicamente acessíveis. Em correntes de blocos federadas, o acesso pode ser público ou restrito aos membros das federações, e varia para cada aplicação [Androulaki et al. 2018, Smolensk 2018]. Em correntes privadas, a leitura só é permitida às entidades autorizadas.

Tabela 1.1. Comparação entre diferentes categorias de correntes de blocos.

	Pública	**Federada**	**Privada**
Permissão de Escrita	Não-permissionada	Permissionada	Permissionada
Permissão de Leitura	Pública	Pública ou Privada	Privada
Modelo de Consenso	Baseado em prova	Baseado em mensagens	Opcional
Incentivo	Obrigatório	Opcional	Opcional
Eficiência	Baixa	Média	Alta
Confiabilidade	Alta	Média	Baixa

- **Modelo de consenso.** Em correntes públicas, o protocolo de consenso considera alterações arbitrárias no conjunto de nós mineradores, resultantes da liberdade de escrita, e implica no uso de protocolos baseados em prova com alto custo computacional [Nakamoto 2008, Wood 2014, Ali et al. 2016]. Em correntes federadas, as restrições de permissão de escrita permitem adotar protocolos de consenso mais eficientes que se baseiam em trocas de mensagens entre mineradores [Alvarenga et al. 2018, Androulaki et al. 2018]. Em correntes privadas, o protocolo de consenso é opcional e as decisões na rede podem ser tomadas de forma centralizada.

- **Incentivo.** Correntes de blocos públicas, devido principalmente ao alto custo dos protocolos de consenso, adotam mecanismos para incentivar nós mineradores a gastar recursos computacionais para propor novos blocos. Em correntes de blocos federadas e privadas, o incentivo é opcional pois os nós mineradores normalmente têm interesse direto na manutenção da corrente de blocos e adotam protocolos menos custosos.

- **Eficiência.** Os protocolos baseados em prova e as grandes quantidades de participantes tornam as correntes de blocos públicas pouco eficientes. Correntes de blocos federadas possuem eficiência maior que as correntes públicas, mas a tomada colaborativa de decisões implica em menor eficiência que as correntes de blocos privadas [Vukolić 2016].

- **Confiabilidade.** A grande descentralização de correntes de blocos públicas dificulta que um atacante domine uma parcela significativa do consenso e confere maior confiabilidade à rede. No entanto, em correntes federadas e principalmente em correntes privadas, a quantidade de mineradores e pode trazer riscos à segurança da rede. Em especial, correntes federadas e privadas são menos tolerantes a ataques de conluio e ataques *Sybil* [Douceur 2002].

Além das características citadas, a anonimidade na rede é uma decisão de projeto fundamental que define a forma de identificar indivíduos na rede. A maior parte das aplicações de correntes de blocos utiliza uma chave pública ou um *hash* assinado como endereço [Androulaki et al. 2018,Nakamoto 2008,Azaria et al. 2016,Ali et al. 2016]. Este tipo de identificação provê autenticidade às transações e garante a pseudo-anonimidade dos usuários. Ainda que a anonimidade não seja totalmente garantida, a maior parte das carteiras automatiza a criação um novo endereço para cada transação realizada, dificultando a associação entre endereço e pessoa física. Para casos em que deseja-se revelar a

identidade de um usuário, é possível associar identificadores a pessoas através de certificados públicos ou dicionários registrados na própria corrente de blocos. Esta identificação é especialmente importante em aplicações que proveem auditabilidade e rastreabilidade de transações [Alvarenga et al. 2018, Bozic et al. 2017, Huh et al. 2017, Lee et al. 2016].

1.3.3. Consenso em Correntes de Blocos

A corrente de blocos é um livro-razão distribuído que reúne uma lista crescente de registros controlada por múltiplas entidades sem garantia de confiança mútua. Os nós mineradores comunicam-se através de uma rede par a par para construir a corrente de blocos de forma colaborativa. No entanto, a conectividade da rede pode ser interrompida e mineradores podem individualmente sofrer falhas desastrosas (*crash failures*), agir de forma maliciosa ou entrar em conluio para controlar parte significativa da rede (falhas bizantinas). É necessário, portanto, adotar um protocolo tolerante a falhas para garantir a consistência do sistema e que todos os mineradores atinjam o mesmo estado global. Assim, a corrente de blocos torna-se uma entidade resiliente que provê segurança, confiabilidade, disponibilidade, auditabilidade e integridade aos usuários.

Consenso é o processo pelo qual um grupo de entidades independentes atingem a mesma decisão de maneira distribuída. Formalmente, todo protocolo de consenso em correntes de blocos deve garantir, sob a presença de falhas, as propriedades:

- **Terminação (*liveness*)**: todo minerador correto [8] eventualmente decide por um bloco B_i a ser inserido na corrente.
- **Acordo (*safety*)**: o bloco B_i de todos os mineradores corretos é idêntico.
- **Validade (*validity*)**: o bloco B_i é o bloco proposto por todos os mineradores corretos no início do consenso.

A propriedade de validade pode ser interpretada como a corretude da decisão, garantindo que o bloco adicionado à corrente é o bloco proposto pelos mineradores corretos. Schneider *et al.* definem dois elementos necessários para obter consenso entre processos distribuídos sob a presença de falhas [Schneider 1993]: i) uma máquina de estados replicada e determinística que armazena um estado global da rede; e ii) um protocolo de consenso que realiza transações de estado de forma descentralizada. Através destes dois elementos, é possível implementar a replicação de máquina de estados (*State Machine Replication* - SMR), uma técnica de tolerância a falhas que garante a consistência de um sistema distribuído.

Os principais desafios de projeto de corrente de blocos com alta disponibilidade são a tolerância a falhas e a coordenação entre os nós mineradores. O teorema CAP (*Consistency*, *Availability*, *Partition*) prova que todo sistema distribuído apresenta um compromisso entre três propriedades [Brewer 2000]:

- **Consistência (*consistency*)**: todos os nós do sistema distribuído possuem os dados mais atuais da rede;

[8] Um minerador correto é um minerador que não está em estado de falha.

- **Disponibilidade (*availability*)**: o sistema está operante, acessível e é capaz de responder corretamente a novas requisições;
- **Tolerância à partição (*partition tolerance*)**: o sistema distribuído opera corretamente mesmo que um grupo de nós falhe. Se for possível haver comportamento malicioso dos nós, o sistema continua funcionando corretamente com alguns nós honestos, mesmo na presença de um grupo de nós maliciosos.

O teorema prova que é impossível para um sistema distribuído atingir plenamente todas as três propriedades [Brewer 2000]. Por isto, na prática, sistemas distribuídos privilegiam uma ou duas propriedades, sacrificando as demais. A replicação contribui para a tolerância a falhas. A consistência é obtida com o uso de algoritmos de consenso que garantem que todos os nós mineradores possuem os mesmos dados. No Bitcoin e em diversas outras aplicações [Nakamoto 2008, Wood 2014, Ali et al. 2016], a consistência é sacrificada em nome da disponibilidade e da tolerância a partições. Isto significa que a consistência não é obtida simultaneamente com as outras duas propriedades, mas sim gradativamente, com o tempo, à medida que os blocos são validados pelos nós da rede. Correntes de blocos baseadas em protocolos tolerantes a falhas bizantinas privilegiam fortemente a consistência, em detrimento da disponibilidade [Schwartz et al. 2014, Androulaki et al. 2018, Buchman 2016, Miller et al. 2016, Sousa et al. 2018].

1.3.3.1. Premissas de um Ambiente de Corrente de Blocos

Um dos pontos mais importantes para a construção de um protocolo de consenso de correntes de blocos são as premissas assumidas em um ambiente distribuído. As premissas devem cobrir todos os elementos essenciais do sistema, como o número máximo de falhas de mineradores, o modelo de sincronia da rede, o modelo de falha dos mineradores, o modelo de atacante, etc. Uma premissa representa uma idealização do mundo real e deve ser avaliada constantemente para cada caso de implementação [Cachin and Vukolić 2017, Vukolić 2016, Zheng et al. 2018]. A seguir, apresenta-se três premissas fundamentais a todo sistema baseado em correntes de blocos.

Uma premissa comum de confiabilidade em uma corrente de blocos com n mineradores independentes é a de que no máximo $f < n/k$ mineradores estarão em estado de falha, onde $k = 2, 3, ..., n$ e os demais $n - f$ mineradores são corretos. Em correntes de blocos públicas, como o Bitcoin e Ethereum [Nakamoto 2008, Wood 2014], assume-se que até 50% da rede pode estar em estado de falha sem comprometer a corretude do sistema. Em implementações federadas e privadas baseadas em protocolos tolerantes a falhas bizantinas [Androulaki et al. 2018, Schwartz et al. 2014, Buchman 2016, Sousa et al. 2018], a tolerância é de até 33% da rede em estado de falha.

O **modelo de falha** de uma corrente de blocos define o tipo de falha que pode ocorrer em nós mineradores. No modelo de falhas desastrosas (*crash failures*), os mineradores podem parar de responder às mensagens do consenso devido a panes, perda de conectividade ou quedas de energia [Ongaro and Ousterhout 2014, Lamport 1998]. No modelo de falhas bizantinas, os mineradores podem, além de não responder, responder às mensagens de forma arbitrária e maliciosa para subverter o sistema [Lamport et al.

1982, Dolev 1982]. No modelo de falhas bizantinas, um processo falho pode exibir qualquer comportamento, incluindo panes, omissão de envios e entregas de mensagens, ou emissão de mensagens falsas de forma proposital. Devido à característica fundamental de desconfiança mútua entre os participantes de uma corrente de blocos, o modelo de falhas bizantinas é amplamente mais utilizado em sistemas baseados em correntes de blocos [Nakamoto 2008, Androulaki et al. 2018, Wood 2014]. O modelo de falha de um sistema de corrente de blocos influencia diretamente no funcionamento do protocolo de consenso a ser adotado.

O **modelo de sincronia** do sistema define o tipo de sincronia existente entre os nós mineradores. No modelo síncrono, assume-se que as mensagens do consenso são sempre entregues sem atraso ou com um atraso bem definido. No modelo eventualmente síncrono [Dwork et al. 1988], as mensagens podem atrasar arbitrariamente, mas chegam ao destino em um tempo finito. No modelo totalmente assíncrono, não há garantia de que as mensagens cheguem ao destino. Não há possibilidade de consenso no modelo assíncrono [Fischer et al. 1985]. As principais propostas de protocolos de consenso em correntes de blocos assumem o modelo eventualmente síncrono devido à sua simplicidade e aplicabilidade ao mundo real [Buchman 2016, Sousa et al. 2018]. Outras propostas utilizam o modelo assíncrono, sem assumir qualquer premissa sobre a infraestrutura de rede [Miller et al. 2016, Duan et al. 2018].

1.3.4. Modelos de Consenso

Um dos principais desafios de consenso em sistemas distribuídos é o resultado de impossibilidade do consenso conhecido como FLP[9]. O resultado prova que, em um sistema com modelo assíncrono e com n participantes identificados, o consenso não possui solução determinística mesmo na presença de uma única falha desastrosa [Fischer et al. 1985]. Assim, para contornar o problema de obtenção de consenso em correntes de blocos, há duas abordagens possíveis: relaxar as garantias do consenso, comprometendo o acordo (*safety*), ou relaxar o modelo de sincronia, assumindo pelo menos o modelo eventualmente síncrono. Como consequência das duas abordagens, o problema do consenso pode ser tratado, respectivamente, de maneira probabilística ou determinística. O consenso probabilístico assume a consistência eventual da rede, isto é, que, na ausência de novas transações, todo participante eventualmente atinge o mesmo estado global [Tseng 2017, Decker et al. 2016]. Isto representa um enfraquecimento da propriedade de acordo (*safety*), uma vez que os participantes podem divergir sobre os dados mais recentes até que a consistência seja atingida. A propriedade de terminação, entretanto, é garantida.

Protocolos de consenso probabilístico baseiam-se em difundir uma decisão local para os vizinhos e, eventualmente, atingir todos os nós mineradores da rede. Em correntes de blocos, isto significa que um bloco é proposto por um nó minerador e adicionado à corrente de blocos antes de ser difundido na rede. Caso o bloco esteja correto, garante-se que os demais nós mineradores eventualmente o validarão e atingirão o mesmo estado global. Consensos probabilísticos possuem como principal vantagem a escalabilidade, uma vez que não é necessário conhecer todos os nós mineradores da rede para se obter consenso. Portanto, este tipo de consenso é mais adequado a correntes de blocos públi-

[9]O teorema recebe esta sigla em homenagem a seus autores: Fisher, Lynch e Patterson.

cas com muitos nós mineradores. Em consensos probabilísticos, dois mineradores podem propor blocos corretos simultaneamente, causando uma bifurcação na corrente de blocos. Nesse caso, todo protocolo de consenso probabilístico deve adotar um algoritmo de desempate para definir uma verdade global. A regra da cadeia mais longa no Bitcoin é a regra de desempate mais conhecida em correntes de blocos [Nakamoto 2008].

Os protocolos de consenso mais comuns em sistemas de consistência eventual são os algoritmos baseados em prova, nos quais um nó minerador que propõe um bloco deve apresentar uma prova de que pode liderar o consenso [Nakamoto 2008, Vasin 2014, Olson et al. 2018]. Nakamoto propôs a prova de trabalho (*Proof of Work* - PoW) como um algoritmo de consenso no qual os mineradores devem provar que gastaram recursos computacionais para resolver independentemente um desafio matemático computacionalmente custoso. Este desafio depende apenas do estado mais recente da corrente de blocos e portanto é totalmente paralelizável. A dificuldade do desafio é ajustada periodicamente de acordo com a capacidade de mineração da rede para garantir uma taxa constante de mineração de blocos. Ao resolver o desafio, o minerador vencedor submete o bloco e a prova de trabalho para todos os seus vizinhos. Qualquer minerador pode tentar gerar um novo bloco a qualquer momento. O nó que vence o desafio recebe incentivos em troca do trabalho e, assim, os nós maliciosos tendem a seguir a ordem instituída pelos nós honestos, pois os ganhos em agir honestamente compensam ações maliciosas [Nakamoto 2008]. A probabilidade de um minerador minerar um bloco é, portanto, proporcional à sua capacidade computacional.

Apesar da inovação, a prova de trabalho do Bitcoin apresenta limitações evidentes. A latência de consenso, de aproximadamente dez minutos, e a vazão de transações, de apenas 7 transações por segundo, consistem em um desafio para atender às necessidades dos sistemas atuais [Popov 2016, Eyal et al. 2016]. Outros algoritmos baseados em prova procuram mitigar o excesso de gasto energético e as limitações de desempenho presentes na prova de trabalho [Vasin 2014, Schwartz et al. 2014, Kiayias et al. 2017]. Uma breve explicação dos algoritmos mais conhecidos é mostrada a seguir.

- **Proof of Stake (PoS).** Em vez de gastar recursos computacionais, um nó minerador deve apostar parte de seus ativos para receber uma chance de minerar um bloco. Sua chance é proporcional à quantia de ativos apostados. Nos últimos anos, a prova de participação têm se destacado devido ao desejo do Ethereum de abandonar a prova de trabalho para implementar PoS [Tikhomirov 2018].

- **Delegated Proof of Stake (DPoS).** Os nós mineradores utilizam seus ativos para eleger delegados em um quórum que define o bloco a ser adicionado. A quantidade de votos de um minerador é proporcional aos seus ativos [Kiayias et al. 2017].

- **Proof of Burn (PoB).** Como o PoS, porém os ativos são imediatamente destruídos [Paul et al. 2014].

- **Proof of Authority (PoA).** Similar ao DPoS, porém o conjunto de delegados (autoridades) é pré-determinado em acordo e suas identidades são públicas e verificáveis por qualquer participante da rede [Angelis et al. 2018].

- **Proof of Capacity (PoC).** A probabilidade de propor um bloco é proporcional ao espaço de armazenamento cedido à rede por um nó minerador. Quanto maior a capacidade de armazenamento em disco, maior o domínio sobre o consenso [Zheng et al. 2018].
- **Proof of Elapsed Time (PoET).** Cada nó minerador recebe um temporizador aleatório decrescente e o nó cujo temporizador terminar primeiro propõe o próximo bloco [Olson et al. 2018]. Este protocolo de consenso funciona exclusivamente em *hardware* que suportam a tecnologia *Intel software guard extensions* (SGX). O Intel SGX garante a distribuição aleatória de temporizadores e que nenhuma entidade tem acesso a mais de um nó minerador.

O consenso tolerante a falhas bizantinas utiliza a primeira abordagem, oferecendo garantias determinísticas e apoiando-se em redes parcialmente síncronas para assegurar o progresso.

Em redes bem-definidas, como redes permissionadas síncronas ou eventualmente síncronas, é possível obter consenso determinístico apoiando-se nas garantias de entrega de mensagens da rede. Neste caso, enquanto a rede não oferece as condições de estabilidade (sincronia) para que haja consenso, a terminação (*liveness*) é comprometida, mas não o acordo (*safety*). Assim, o consenso determinístico sempre assegura a consistência do sistema, possivelmente sacrificando seu progresso.

O consenso determinístico é alcançado através de protocolos de consenso baseados em quórum. Neste tipo de consenso, todos os nós mineradores devem votar pela aprovação ou rejeição de um bloco e atingir um consenso antes de adicionar o novo bloco à corrente. Como consequência, todos os nós mineradores devem ser conhecidos, identificados e deve haver sincronia entre os nós para a troca de mensagens. O consenso baseado em quórum garante que a adição de um bloco à corrente ocorre ao mesmo tempo para todas as réplicas. Portanto, a consistência do sistema é garantida em qualquer momento e não existem bifurcações na corrente de blocos. A tolerância a falhas e comportamento malicioso é definida de acordo com as especificações de cada protocolo de consenso. Um consenso determinístico baseado em quórum é mais eficiente que algoritmos baseados em prova [Schwartz et al. 2014, Dinh et al. 2017].

Uma classe de protocolos de replicação de máquina de estados particularmente interessante para as correntes de blocos é a de protocolos tolerantes a falhas bizantinas (*Byzantine Fault Tolerant* - BFT), que garantem a consistência da corrente de blocos sob a presença de comportamento malicioso [Lamport et al. 1982]. Protocolos BFT são capazes de atingir até dezenas de milhares de transações por segundo com baixa latência, sob a restrição de operarem em redes com sincronia eventual e com poucos nós mineradores [Cachin and Vukolić 2017, Vukolić 2016]. Isto faz com que os protocolos BFT sejam ideais para redes permissionadas nas quais os participantes podem agir de maneira maliciosa. Em especial, a tolerância a falhas bizantinas assume as condições propostas no problema dos generais bizantinos, descrito por Lamport [Lamport et al. 1982]:

- todos os participantes são conhecidos e identificados[10];

[10]Existem trabalhos que estudam o consenso bizantino em que os participantes são desconhecidos [Al-

- todos os participantes podem trocar mensagens diretamente;
- as mensagens trocadas chegam ao destino em tempo finito;
- toda mensagem é assinada por seu emissor;
- todo participante pode falhar, incluindo o líder do consenso;
- participantes em falha podem mentir ou omitir mensagens;
- as mensagens podem ser alteradas por um terceiro no meio do caminho;
- as mensagens podem ser perdidas, reordenadas ou duplicadas;
- o caminho entre dois participantes pode ser interrompido;

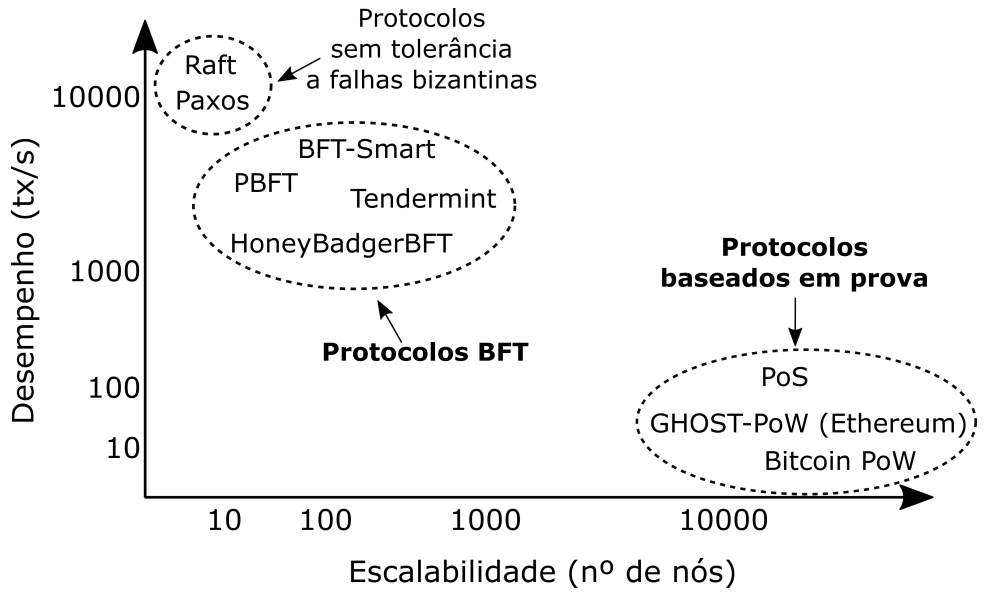


Figura 1.8. Comparação de desempenho e escalabilidade de protocolos sem tolerância a falhas bizantinas, protocolos BFT e protocolos baseados em prova. Os protocolos BFT sacrificam escalabilidade para tolerar falhas bizantinas com maior desempenho em redes permissionadas.

A ideia principal dos protocolos BFT é eleger um líder que inicia e comanda uma rodada de consenso distribuindo o novo bloco proposto a todos os participantes do consenso. Os participantes do consenso respondem ao líder com suas avaliações e as difundem para todos os outros participantes, para gerar provas coletivas da avaliação e prevenir uma possível ação maliciosa do líder. Ao receber uma quantidade suficiente de votos positivos, cada participante considera que o quórum foi alcançado e os participantes escrevem o novo bloco na corrente simultaneamente. Protocolos BFT toleram no máximo $f = \frac{n}{3} + 1$ falhas de participantes. Alguns protocolos toleram menos falhas devido a decisões de projeto [Schwartz et al. 2014, Buchman 2016].

chieri et al. 2018]

Protocolos tolerantes a falhas bizantinas representam um meio termo entre um ambiente totalmente sem confiança e um ambiente totalmente confiável. Os protocolos BFT promovem uma camada de confiança a mais em comparação a protocolos tolerantes apenas a falhas desastrosas [Ongaro and Ousterhout 2014, Lamport 1998] pois toleram comportamento malicioso na rede. No entanto, toleram menos falhas do que os clássicos 50% dos protocolos baseados em prova [Nakamoto 2008].

A Figura 1.8 apresenta uma comparação em alto nível dos principais protocolos de consenso adotados em sistemas baseados em livro-razão distribuído [Mingxiao et al. 2017, Han et al. 2018, Santos and Schiper 2012, Buchman 2016, Bessani et al. 2014, Vukolić 2016, Cachin and Vukolić 2017]. A eventual consistência dos protocolos baseados em prova provê a escalabilidade necessária para as redes públicas, em detrimento da eficiência. Os protocolos BFT proveem alto desempenho para ambientes com número limitado e conhecido de nós. A eficiência e escalabilidade de cada classe de consenso está diretamente associada às suas premissas. Os protocolos baseados em prova são os mais seguros, escaláveis e menos eficientes. Os protocolos BFT e tolerantes a falhas desastrosas são mais eficientes e menos seguros. A análise minuciosa do modelo de segurança e requisitos de eficiência da rede, portanto, é fundamental para a escolha do protocolo de consenso.

1.4. Fatiamento Seguro de Redes Através de Corrente de Blocos

Esta seção mostra um estudo de caso de aplicação da tecnologia de corrente de blocos para prover segurança em um ambiente com virtualização de funções de rede multi-inquilino e multiusuário que não possuem confiança entre si.

As redes móveis de próxima geração fornecem um modelo de conectividade com múltiplos serviços de rede adaptados para atender a demanda de cada segmento de cliente. As redes definidas por *software* (*Software-Defined Networking* - SDN) e a virtualização de funções de rede (*Network Function Virtualization* - NFV) são as principais tecnologias que utilizam a virtualização para fornecer agilidade, automação e capacidade de programação da rede. Assim, as tecnologias NFV e SDN criam uma cadeia de funções de rede (*Service Function Chain* - SFC) fim-a-fim [Halpern and Pignataro 2015] para fornecer serviços sob demanda e adaptados a cada aplicação. Apesar de a associação de NFV e SDN fornecer a agilidade e o baixo custo desejados pelas telecomunicações, surgem novos desafios de segurança [Medhat et al. 2017]. Além disso, o impacto de possíveis ataques aumenta, pois ataques a hospedeiros de funções de rede podem comprometer simultaneamente milhares de usuários [Bhamare et al. 2016]. Portanto, é de grande importância reduzir os possíveis vetores de ataque a funções virtuais de rede (*Virtual Network Functions* - VNF) e fornecer um gerenciamento de configuração seguro e confiável. Garantir o isolamento entre fatias de rede é essencial para evitar ataques comuns em infraestruturas compartilhadas. Além disso, os inquilinos de cada fatia compartilham a mesma infraestrutura de nuvem, e as cadeias podem envolver funções virtualizadas instanciadas em domínios de provedores concorrentes. O ambiente multi-inquilino e multi-domínio aumenta a possibilidade de ataques dentro da nuvem, ao mesmo tempo que dificulta a responsabilização dos provedores de serviços quando ocorre uma falha. É necessário garantir que a cadeia de serviços seja construída de maneira confiável em um ambiente sem confiança entre os pares. Logo, a capacidade de auditoria é obrigatória para identificar

uma configuração de VNF defeituosa ou comprometida, e a tecnologia de corrente de blocos fornece as características necessárias de não repúdio e imutabilidade do histórico de configuração de uma função virtual.

Rebello *et al.* propõem uma arquitetura na qual cada fatia de rede baseada em corrente de blocos aborda um ou mais casos de uso do 5G, criando redes isoladas com segurança e confiança [Rebello et al. 2019b]. A Figura 1.9 descreve um cenário que usa corrente de blocos para três fatias de rede: uma fatia de rede móvel, uma fatia de indústria 4.0 e uma fatia de redes veiculares. Para garantir a justiça no protocolo de consenso, cada centro de dados pode hospedar no máximo um nó de corrente de blocos por fatia de rede. Os nós de corrente de blocos em uma fatia são invisíveis para qualquer entidade fora da fatia.

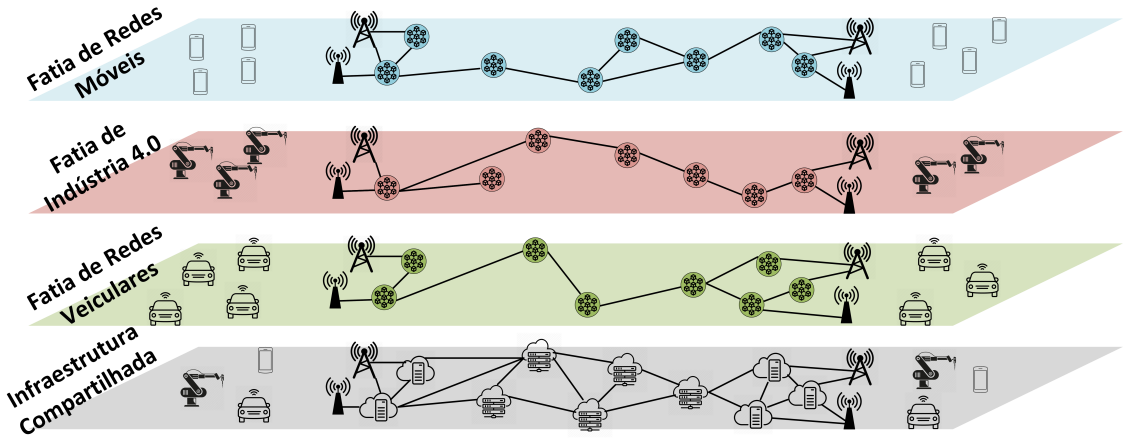


Figura 1.9. Fatias de rede isoladas através de corrente de blocos em uma infraestrutura física compartilhada. Cada fatia de rede é adaptada às necessidades de um caso de uso.

A arquitetura apresentada fornece a capacidade de auditoria de criação e gerenciamento de fatias, registrando todas as operações de orquestração da VNF em uma corrente de blocos de gerenciamento. A corrente de blocos de gerenciamento registra as operações de orquestração que criam ou modificam uma fatia de rede. Toda operação é assinada pelo cliente que solicitou a modificação. Os participantes da corrente de blocos devem validar cada operação por consenso e fornecer uma prova assinada irrefutável de que a transação foi aceita antes que as operações sejam realizadas. A solicitação assinada combinada com o registro permanente fornecido pela corrente de blocos garante que um comportamento malicioso seja rastreável. Assim, o emprego de uma corrente de blocos gerencial garante a procedência, a prestação de contas e a rastreabilidade das falhas em um ambiente NFV multi-inquilino e multi-domínio. Além disso, a corrente de blocos pode se servir de contratos inteligentes para prover automação e transparência em ambientes sem confiança distribuídos, em vez de confiar em um determinado nó para receber e processar transações. As propriedades de automação e transparência dos contratos inteligentes são ideais para criar fatias de rede fim-a-fim seguras que envolvem VNFs em vários domínios concorrentes, pois garantem que todos os nós da rede obedeçam ao mesmo conjunto de regras e que o código executado seja visível para qualquer nó participante.

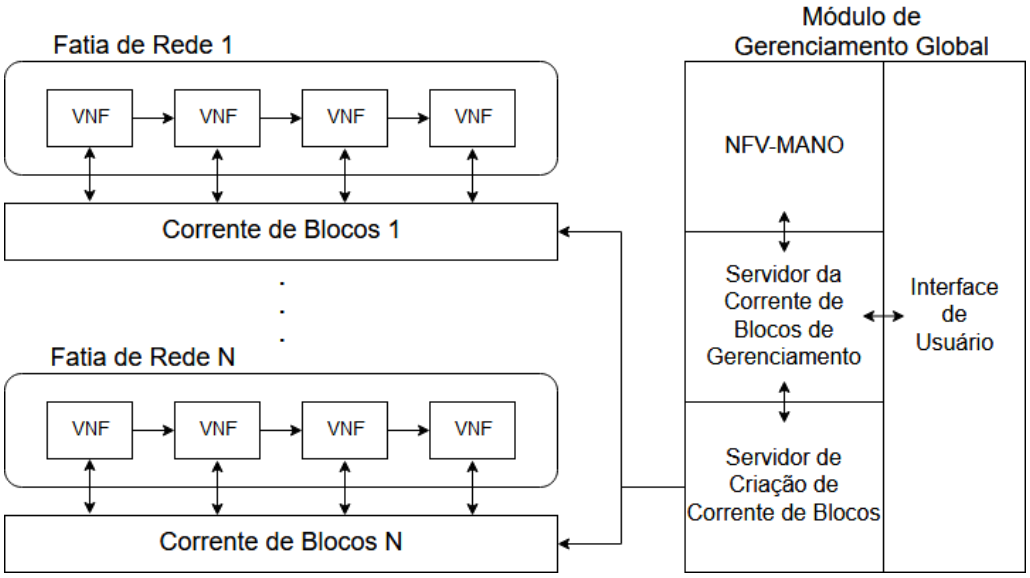


Figura 1.10. A arquitetura proposta por Rebello *et al.* baseada em corrente de blocos para fatiamento de rede. O usuário interage com o módulo de gerenciamento global para criar fatias de rede seguras. Cada VNF em uma fatia de rede é conectada a uma corrente de blocos responsável por registrar solicitações de configuração e informações relevantes, conforme especificado pelo usuário.

A Figura 1.10 mostra os quatro componentes principais: uma interface de usuário, o módulo de gerenciamento e orquestração NFV (NFV-MANO), um módulo de servidor de criação de corrente de blocos e um módulo de corrente de blocos de gerenciamento. Os módulos compõem o módulo de gerenciamento global, que é responsável por conectar o cliente aos serviços oferecidos. Nesta arquitetura, o cliente interage com o módulo de gerenciamento global por meio de uma interface de usuário para criar/modificar uma fatia de rede segura ou para solicitar informações de fatia, como configurações de VNFs e cadeias de funções. O cliente especifica as características da fatia, como VNFs desejadas e restrições de posicionamento, e a categoria de corrente de blocos correspondente. O módulo de interface do usuário converte as especificações em operações de criação de fatias/corrente de blocos e as envia como transações assinadas para aprovação na corrente de blocos de gerenciamento. Depois que as transações são aprovadas, o módulo NFV-MANO e o módulo de criação de corrente de blocos verificam a corrente de blocos de gerenciamento para obter operações pendentes. Os módulos executam as operações para criar novas fatias seguras e emitem transações de resposta assinadas para a corrente de blocos de gerenciamento. O cliente pode então interagir com o módulo de interface do usuário para verificar se a fatia segura solicitada foi criada com sucesso.

1.5. Atividade Prática (*Hands-On*): Desenvolvimento de uma Corrente de Blocos para Telecomunicações

O objetivo da atividade prática é evidenciar a segurança proporcionada pelo uso de corrente de blocos em um ambiente de funções virtualizadas de rede, registrando operações de criação e configuração de VNFs, garantindo confiabilidade, integridade e auditabilidade das informações armazenadas. A atividade prática utiliza, como estudo de

caso, a arquitetura de fatiamento de rede apresentada na seção anterior com dois tipos de corrente de blocos. A demonstração usa a plataforma *Hyperledger Fabric* para a criação de uma corrente de blocos permissionada. O *Hyperledger Fabric* é uma plataforma de código aberto para o desenvolvimento de correntes de blocos permissionadas em um ambiente sem confiança entre os participantes.

Para realizar esta atividade, é necessário uma máquina com acesso à Internet que possua pelo menos 4 GB de memória RAM e processador Intel Core i3 ou equivalente. A arquitetura do sistema deve ser de 64 bits e recomenda-se o uso de um sistema operacional baseado em Linux por experiência de uso, maior facilidade para programação e por serem baseados em código aberto e gratuito. São disponibilizados dispositivos USB removíveis (*pen drives*) com imagens para criação de uma máquina virtual com sistema operacional *Debian 9.0 (Stretch)* que inclui todos os pré-requisitos necessários para utilização do Hyperledger Fabric já instalados. Vale ressaltar que o computador utilizado pelo participante deve possuir uma ferramenta de virtualização como o VirtualBox[11] ou VMWare Workstation Player[12] para utilizar a máquina virtual fornecida pelos autores. Para dar maior dinamicidade à aula prática, os participantes que possuem acesso à Internet também podem baixar e instalar os pacotes necessários manualmente enquanto os demais participantes obtêm a imagem dos dispositivos removíveis. Os pré-requisitos estão listados na documentação do Hyperledger Fabric[13].

1.5.1. Auditoria de Criação, Leitura, Atualização e Remoção de Funções de Rede

A programabilidade do núcleo da rede expõe as funções de redes a um alto número de ameaças. Em um cenário multi-inquilino e multi-domínio, as ameaças podem afetar o tráfego que flui por uma VNF afetada. Desta maneira, a identificação de funções comprometidas de rede é essencial para garantir a segurança de um ambiente de funções virtualizadas de rede. O uso de corrente de blocos permite o registro permanente de operações envolvendo funções de rede, garantindo a auditabilidade do histórico de operações. Além disso, o uso de criptografia de chaves assimétricas garante a identificação e facilita a responsabilização dos envolvidos no comprometimento da função.

1.5.1.1. A Plataforma Hyperledger Fabric

O Hyperledger[14] é uma iniciativa colaborativa e de código aberto da Linux Foundation que objetiva desenvolver tecnologias baseadas em corrente de blocos para a indústria. A colaboração global inclui empresas de tecnologia da informação, mercado financeiro, Internet das Coisas, cadeias de suprimentos, saúde e financiamento.

Um dos diversos projetos inclusos na iniciativa Hyperledger é o *Hyperledger Fabric* [Androulaki et al. 2018], uma plataforma proposta e desenvolvida pela IBM para a implementação de redes permissionadas baseadas em corrente de blocos. O Hyperledger Fabric é o projeto mais maduro da iniciativa Hyperledger e o mais utilizado por empre-

[11]Disponível em https://www.virtualbox.org/
[12]Disponível em https://www.vmware.com/br/products/workstation-player.html
[13]Disponível em https://hyperledger-fabric.readthedocs.io/en/release-1.4/prereqs.html
[14]Disponível em https://www.hyperledger.org/

sas que desenvolvem aplicações privadas baseadas em corrente de blocos. A arquitetura modular e plugável do Fabric permite o uso de diferentes tipos de protocolos de consenso, diferentes linguagens de programação para a criação de contratos inteligentes e diferentes bancos de dados para armazenar a corrente de blocos e seu estado atual. Outra proposta do Hyperledger Fabric é criar redes nas quais podem coexistir diversas correntes de blocos isoladas, independentes, e com diferentes configurações, contratos inteligentes e participantes. As entidades que participam de uma rede baseada em correntes de blocos do Fabric podem ter diferentes funções e permissões de acesso à corrente de blocos. As entidades com poder administrativo podem modificar as permissões de escrita e leitura das demais entidades em cada corrente de blocos através de transações de configuração, enquanto entidades comuns têm permissão para emitir transações que não sejam sensíveis ao funcionamento da rede, como uma transferência de recursos entre duas entidades. Há a possibilidade de definir políticas específicas e adaptadas para a validação de transações que exigem a assinatura por múltiplos validadores. Os desenvolvedores das correntes de blocos no Fabric podem configurar permissões de leitura e escrita para criar redes permissionadas, nas quais todos os nós da rede se conhecem. Isto é ideal para ambientes empresariais ou de consórcio, tipicamente constituídos por até dezenas de nós. O Hyperledger Fabric fornece um serviço de identidade e associação de membros que gerencia os identificadores dos usuários, e autentica todos os participantes na rede automaticamente. Além disso, podem existir nós internos em cada organização/empresa. Todas essas configurações disponíveis no Hyperledger Fabric são adequadas e facilitam a implementação de correntes de blocos adaptadas a cada aplicação, com necessidades e características diferentes, em um ambiente empresarial.

Nós e canais são os conceitos chave mais importantes de uma rede baseada em corrente de blocos permissionada do Hyperledger Fabric. Os nós representam as entidades que participam do processamento de uma transação ou mantêm uma cópia da corrente de blocos. O Hyperledger Fabric provê três tipos de nós: clientes, pares (*peers*) e ordenadores. Um cliente representa um usuário que envia transações aos pares para validação e assinatura, e transmite propostas de transação assinadas para o serviço de ordenação. Os pares são um elemento fundamental da rede porque executam propostas de transação, validam transações e mantém os registros na corrente de blocos. Os pares também instanciam contratos inteligentes e armazenam o estado global, uma representação sucinta do estado mais recente da corrente de blocos. Os nós ordenadores formam coletivamente o serviço de ordenação, que é responsável por estabelecer a ordem total e o empacotamento de todas as transações em um bloco usando um protocolo de consenso. Os ordenadores não participam da execução da transação nem validam transações. O desacoplamento das funcionalidades de ordenação e validação aumenta a eficiência da rede, pois permite o processamento paralelo de cada fase. A Figura 1.11 descreve um exemplo de uma corrente de blocos permissionada com quatro organizações no Hyperledger Fabric. Cada organização recebe transações de clientes e as retransmite para os ordenadores após a validação pelos pares. Cada organização possui um único ordenador, garantindo a justiça no protocolo de consenso.

Caminhos de mensagens diferentes, chamados canais, isolam as correntes de blocos. Um canal Hyperledger Fabric é uma sub-rede de comunicação privada e isolada entre um subconjunto de nós da rede específicos para fornecer privacidade e confidencialidade

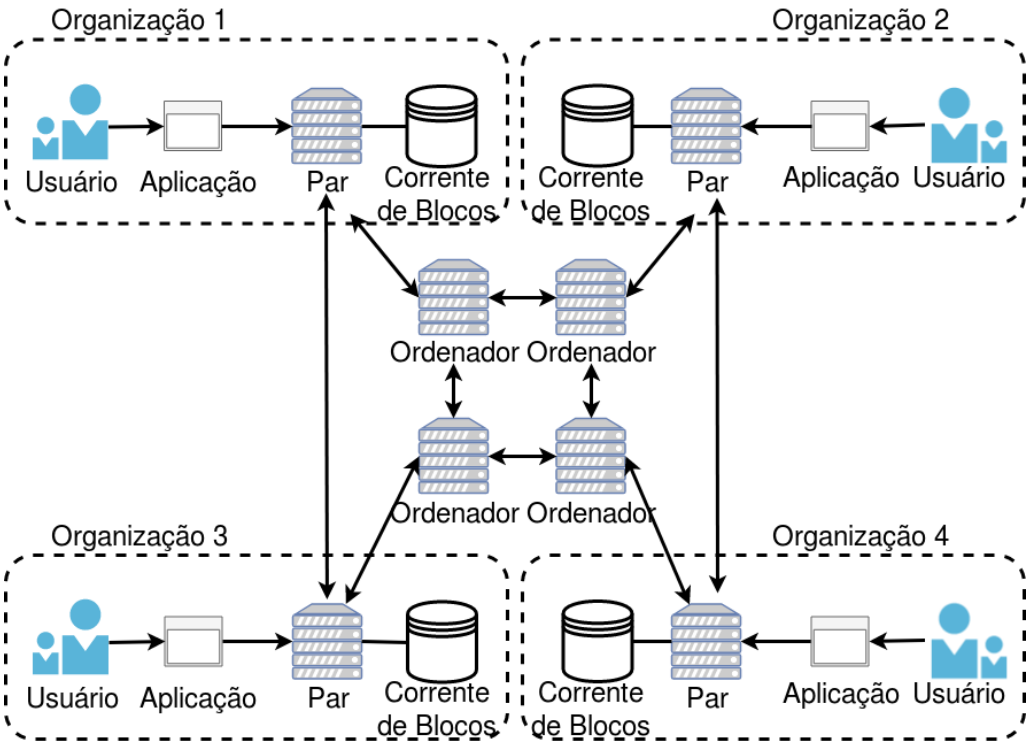


Figura 1.11. Uma corrente de blocos permissionada do Hyperledger Fabric. Usuários de cada organização usam aplicações para emitir transações assinadas e as submetem para validação nos pares. Os pares validam a transação e respondem à aplicação, que retransmite a transação validada para os ordenadores. Os ordenadores trocam mensagens para definir a ordenação global das transações recebidas em um intervalo de tempo e as adicionam em um novo bloco. Depois de um bloco ser proposto e aceito pelos ordenadores através do consenso, os pares atualizam a corrente de blocos e o estado global da rede.

às transações. Todos os dados transmitidos em um canal, incluindo transações, contratos inteligentes, configurações de associação e informações de canal, são invisíveis e inacessíveis a qualquer entidade externa a um canal. As mensagens trocadas em um canal são criptografadas. A funcionalidade do canal é ideal para a proposta de oferecer correntes de blocos personalizadas para diferentes serviços de rede, pois permite que os administradores dos canais estabeleçam diferentes formatos de bloco e transação, além do protocolo de consenso, para cada canal. Portanto, é possível usar canais para oferecer fatias de rede protegidas por correntes de blocos configuradas de forma específica. Formatos de transação são definidos em contratos inteligentes, chamados de *chaincode* no Hyperledger Fabric, escritos em Go, Node.js ou linguagem Java.

1.5.1.2. Exemplo de Desenvolvimento de uma Corrente de Blocos

Esta aula prática desenvolve um protótipo de aplicação que usa a plataforma Hyperledger Fabric [Androulaki et al. 2018] para implementar correntes de blocos entre organizações em ambientes sem confiança. Cada organização mantém uma réplica da corrente de blocos e pode acrescentar blocos através de um protocolo de consenso. O

```
struct instructionTransaction
{
    command                 string
    transactionType         string
    transactionName         string
    issuer                  string
}
initialize queue
initInstruction (instruction <command,name,issuer>)
{
    if instruction is not unique or instruction is not well-formatted:
        return error
    putState (instruction.name, instruction)
    put (transactionID, queue)
    notify orchestrator
    return success
}
```

Lista 1.1. Parte do pseudocódigo do contrato inteligente que emite transações de instrução. O campo de comando contém a operação de orquestração a ser executada por um orquestrador. O contrato estabelece uma fila de instruções pendentes a serem processadas pelo orquestrador.

protótipo implementado segue a arquitetura de fatiamento de rede apresentada na seção anterior. O protótipo implementa cada nó da rede do Hyperledger Fabric como um contêiner em um único computador e envia transações simultaneamente. A aula prática implementa dois contratos inteligentes[15] escritos em Go, que são executados em todos os nós da rede [Alvarenga et al. 2018, Rebello et al. 2019a].

O primeiro contrato inteligente, parcialmente descrito na Lista 1.1, gerencia autonomamente o gerenciamento e a orquestração de VNF através de transações de instrução e resposta. Quando um cliente solicita uma fatia, o servidor da corrente de blocos de gerenciamento emite uma transação de instrução com o comando de instrução. O contrato coloca a transação de instrução em uma fila de transações pendentes de instrução. O código notifica o módulo NFV-MANO, que executa a instrução pendente e envia a saída do comando para o servidor da corrente de blocos de gerenciamento. O módulo NFV-MANO emite uma transação de resposta que inclui um campo contendo o identificador da transação de instrução correspondente. Isso fornece a rastreabilidade de cada transação executada na corrente de blocos e, portanto, possibilita a responsabilização de entidades maliciosas.

O segundo contrato inteligente, descrito na Lista 1.2, define e atualiza a configuração de uma VNF. Um cliente emite uma transação para a corrente de blocos conectada a cada VNF em uma fatia de rede. A transação contém um texto descritivo com a configuração associada no campo de descrição, assim como os dados de configuração no campo de configuração.

O protótipo usa os certificados de autoridade (*Certificate Authorities* - CA) do Hyperledger Fabric para criar e gerenciar certificados digitais em cada nó da rede de corrente

[15] O código completo está disponível em https://github.com/gta-ufrj/hpsr-smart-contracts

```
struct configurationTransaction
{
    configurationIdentifier string
    versionIdentifier       string
    description             string
    configuration           string
    transactionType         string
    transactionName         string
    issuer                  string
}
initConfiguration (configuration <description,configuration,name,issuer
    >){
    if configuration is not unique or configuration is not well-formed:
        return error
    putState (configuration.name, configuration)
    return success
}
```

Lista 1.2. Pseudocódigo parcial para emitir transações de configuração. O campo configurationIdentifier contêm um identificador único para a configuração.

de blocos. Certificados digitais garantem auditabilidade e que somente nós certificados e autorizados podem participar da rede de corrente de blocos.

Este parágrafo inicia um roteiro a ser executado pelos participantes desta aula prática realizarem a atividade prática descrita. A atividade prática foi desenvolvida utilizando a versão 1.4.0 do Hyperledger Fabric. Os comandos de versões anteriores ou posteriores podem ser incompatíveis com a versão utilizada nesta atividade. Para baixar os arquivos necessários, os participantes da aula prática devem executar os seguintes comandos:

```
git clone https://github.com/keenkit/fabric-sample-
with-kafka.git
```

```
cd fabric-sample-with-kafka
```

```
sed -i 's/1.2.0/1.4.0/' get-byfn.sh && ./get-byfn.sh
```

```
cd first-network
```

```
curl -sSL http://bit.ly/2ysbOFE | bash -s 1.4.0 1.4.0
0.4.14
```

O participante deve criar o diretório `contract` dentro do diretório `examples/chaincode` e baixar o contrato inteligente que será utilizado ao longo desta atividade. De dentro do diretório `first-network`, o participante deve executar:

```
cd examples/chaincode && mkdir contract && cd contract
```

```
wget https://github.com/gta-ufrj/hpsr-smart-contracts/raw/
master/contract.go
```

O arquivo `crypto-config.yaml` contém as informações referentes aos participantes iniciais da rede. As configurações presentes nesse arquivo informam os endereços de cada nó da rede, o número de ordenadores, as organizações participantes da rede e

a quantidade de participantes por organização. Esses dados são utilizados pela ferramenta `cryptogen` para gerar pares de chaves assimétricas e certificados para os nós, que serão utilizados no processo de validação, controle de acesso e não-repúdio às transações realizadas por um membro da rede. Também é necessário o arquivo `configtx.yaml` para configuração da rede. Nele estão contidas informações sensíveis à rede como o tipo de consenso utilizado, tempo máximo para a criação de um novo bloco, tamanho máximo das transações contidas em um bloco, tamanho máximo em *bytes* do bloco, políticas para validação de transações, e perfil de cada nó. A ferramenta `configtxgen` utiliza o arquivo `configtx.yaml` para criar o bloco gênesis que contém as informações de configuração da rede.

De dentro do diretório `first-network`, o usuário deve gerar os certificados que serão usados na rede utilizando o comando:

```
../bin/cryptogen generate --config=<caminho do arquivo de
configuração>
```

- caminho do arquivo de configuração: ./crypto-config.yaml

O comando cria um diretório chamado `crypto-config` que contém os certificados e chaves da rede. Em seguida, o usuário deve indicar o diretório do arquivo de configuração `configtx.yaml` para a ferramenta `configtxgen`. Feito isso, utilize a ferramenta `configtxgen` para gerar o bloco gênesis no diretório `channel-artifacts`. Esses dois passos são realizados da seguinte forma:

```
export FABRIC_CFG_PATH=$PWD

../bin/configtxgen - profile <perfil do canal>
-channelID <nome do canal> -outputBlock
./channel-artifacts/genesis.block
```

- perfil do canal: TwoOrgsOrdererGenesis
- nome do canal: byfn-sys-channel

Para a criação do canal, o seguinte comando deve ser executado:

```
export CHANNEL_NAME=<nome do canal> &&
../bin/configtxgen -profile <perfil do canal>
-outputCreateChannelTx ./channel-artifacts/channel.tx
-channelID $CHANNEL_NAME
```

- perfil do canal: TwoOrgsChannel
- nome do canal: sbrcchannel[16]

O comando cria um arquivo chamado `channel.tx` contendo as configurações do canal dentro do diretório `channel-artifacts`.

[16]O nome do canal deve conter apenas letras minúsculas.

Pares-âncora (*anchor peers*) conectam uma organização a outra. Pares de uma organização comunicam com os pares-âncora para descobrir pares de outras organizações. Cada organização possui ao menos um par-âncora. O comando para a definição do par-âncora para as organizações configuradas deve ser executado para cada organização listadas nos arquivos de configuração:

```
../bin/configtxgen -profile <perfil do canal>
-outputAnchorPeersUpdate ./channel-artifacts/<nome da
organização>anchors.tx -channelID $CHANNEL_NAME -asOrg
<nome da organização>
```

- perfil do canal: TwoOrgsChannel
- nome da organização: Org1MSP

Como o exemplo desta atividade usa duas organizações:

```
../bin/configtxgen -profile <perfil do canal>
-outputAnchorPeersUpdate ./channel-artifacts/<nome da
organização>anchors.tx -channelID $CHANNEL_NAME -asOrg
<nome da organização>
```

- perfil do canal: TwoOrgsChannel
- nome da organização: Org2MSP

A rede de corrente de blocos do Hyperledger Fabric usa contêineres como nós. O usuário deve usar um arquivo de configuração em conjunto com a ferramenta *Docker Compose* para criar a rede. A plataforma Fabric disponibiliza o arquivo de configuração `docker-compose-kafka.yaml` para facilitar a configuração e instanciação de contêineres.

```
IMAGE_TAG=latest docker-compose -f <arquivo do Docker
Compose> up -d
```

- arquivo do Docker Compose: docker-compose-kafka.yaml

Para acessar o contêiner cliente:

```
docker exec -it cli bash
```

Agora, o usuário deve passar as configurações do canal criada em um dos passos anteriores para criar o canal do Hyperledger Fabric:

```
export CHANNEL_NAME=sbrcchannel
```

```
peer channel create -o <nome do ordenador>:<porta do
ordenador> -c $CHANNEL_NAME -f ./channel-artifacts/
channel.tx --tls --cafile <caminho do certificado>
```

- nome do ordenador: orderer0.example.com

- porta do ordenador: 7050
- caminho do certificado: /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto /ordererOrganizations/example.com/orderers/orderer0.example.com /msp/tlscacerts/tlsca.example.com-cert.pem

Para inicializar um par no canal criado, os usuários devem executar os seguintes comando:

```
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/
hyperledger/fabric/peer/crypto/peerOrganizations/
org1.example.com/users/Admin@org1.example.com/msp

export CORE_PEER_ADDRESS=peer0.org1.example.com:7051

export CORE_PEER_LOCALMSPID="Org1MSP"

export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/
github.com/hyperledger/fabric/peer/crypto/peerOrganizations
/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

peer channel join -b sbrcchannel.block
```

Os participantes devem repetir os cinco comandos acima mudando o par e a organização. Como o exemplo desta atividade usa quatro pares, os comandos devem ser modificados para atender aos peer0 e peer1 da organização Org1 e aos peer0 e peer1 da organização Org2.

Com todos os pares adicionados, é necessário atualizar o canal para escolher os pares-âncoras de cada organização. Uma atualização no canal do Hyperledger Fabric adiciona uma informação de configuração do canal. Neste exemplo, os pares-âncoras são o peer0 da organização Org1 e peer0 da organização Org2.

```
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/
hyperledger/fabric/peer/crypto/peerOrganizations/
org1.example.com/users/Admin@org1.example.com/msp

export CORE_PEER_ADDRESS=peer0.org1.example.com:7051

export CORE_PEER_LOCALMSPID="Org1MSP"

export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/
github.com/hyperledger/fabric/peer/crypto/peerOrganizations
/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

peer channel update -o <ordenador> -c
$CHANNEL_NAME -f ./channel-artifacts/<org name>anchors.tx
-tls -cafile <caminho do certificado>
```

- ordenador: orderer0.example.com:7050
- caminho do certificado: /opt/gopath/src/github.com/hyperledger/fabric/peer /crypto/ordererOrganizations/example.com/orderers/orderer0.example.com /msp/tlscacerts/tlsca.example.com-cert.pem

Os participantes devem repetir os cinco comandos acima para atender ao peer0 da organização Org2.

Agora, é necessário instalar o contrato em cada par que executa e apoia as transações.

```
export CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/
hyperledger/fabric/peer/crypto/peerOrganizations/
org1.example.com/users/Admin@org1.example.com/msp

export CORE_PEER_ADDRESS=peer0.org1.example.com:7051

export CORE_PEER_LOCALMSPID="Org1MSP"

export CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/
github.com/hyperledger/fabric/peer/crypto/peerOrganizations
/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

go get -u github.com/golang-collections/go-datastructures
/queue && peer chaincode install -n <nome do chaincode>
-v 1.0 -p <diretório do chaincode>
```

- nome do chaincode: mycc
- diretório raiz do chaincode: github.com/chaincode/contract

Os participantes devem repetir os cinco comandos acima mudando o par e a organização. Como o exemplo desta atividade usa quatro pares, os comandos devem ser modificados para atender aos peer0 e peer1 da organização Org1 e aos peer0 e peer1 da organização Org2.

Em seguida, o participante deve instanciar o *chaincode* no canal e também definir a política de endosso (*endorsement*) do canal:

```
peer chaincode instantiate -o <ordenador>
--tls --cafile <caminho do certificado> -C $CHANNEL_NAME
-n mycc -v 1.0 -c <mensagem em JSON para o chaincode>
-P <política do canal>
```

- ordenador: orderer0.example.com:7050
- caminho do certificado: /opt/gopath/src/github.com/hyperledger/fabric/peer /crypto/ordererOrganizations/example.com/orderers/orderer0.example.com /msp/tlscacerts/tlsca.example.com-cert.pem
- mensagem em JSON para o chaincode: ‘{"Args":["init"]}’
- política do canal: "AND (‘Org1MSP.peer’,‘Org2MSP.peer’)"

Neste caso, a política descrita acima define que uma transação efetuada no canal deve ser endossada por um par pertencente à organização 1 e por um par pertencente à organização 2.

Para gerar uma transação, o Fabric disponibiliza o comando `invoke`, que chama uma função do contrato instanciado no canal. O comando recebe como argumento o endereço e o caminho dos certificados dos pares, o serviço de ordenação, o nome do canal, o nome do contrato instanciado e a mensagem para o contrato em formato JSON. Como exemplo, o comando abaixo emite uma transação na rede.

```
peer chaincode invoke -o <ordenador>
--tls --cafile <caminho do certificado> -C $CHANNEL_NAME
-n mycc --peerAddresses <endereço do par de org1>
--tlsRootCertFiles <caminho do certificado do par>
--peerAddresses <endereço do par de org2>
--tlsRootCertFiles <caminho do certificado do par>
-c <mensagem em JSON para o chaincode>
```

- ordenador: orderer0.example.com:7050
- caminho do certificado: /opt/gopath/src/github.com/hyperledger/fabric/peer /crypto/ordererOrganizations/example.com/orderers/orderer.example.com /msp/tlscacerts/tlsca.example.com-cert.pem
- endereço do par de org1: peer0.org1.example.com:7051
- caminho do certificado do par: /opt/gopath/src/github.com/hyperledger/fabric/peer /crypto/peerOrganizations/org1.example.com/peers /peer0.org1.example.com/tls/ca.crt
- endereço do par de org2: peer0.org2.example.com:7051
- caminho do certificado do par: /opt/gopath/src/github.com/hyperledger/fabric /peer/crypto/peerOrganizations/org2.example.com/peers /peer0.org2.example.com/tls/ca.crt
- mensagem em JSON para o chaincode: ‘{"Args":["initInstructionTransaction", "transaction","issuer","instruction"]}’

O Fabric permite serviços de busca e pesquisa de histórico de transações através do comando `peer chaincode query`. O comando recebe como argumento o nome do canal, o nome do contrato instanciado e a mensagem para o contrato em formato JSON. Um exemplo:

```
peer chaincode query -C $CHANNEL_NAME -n mycc -c
<chaincode JSON message>
```

- mensagem em JSON para o chaincode: ‘{"Args":["getHistoryForTransaction" , "transaction"]}’

1.6. Considerações Finais, Perspectivas e Problemas em Aberto

A tecnologia de virtualização de funções de rede permite diferenciar serviços encadeando funções virtualizadas entre diferentes provedores de infraestruturas de nuvem sem garantia de confiança entre os pares. Cada operadora possui um centro de dados, que contém um orquestrador de funções virtualizadas de rede (*Virtual Network Functions* - VNF) e uma infraestrutura de virtualização baseada em máquinas de propósito geral para hospedar máquinas virtuais e VNFs. A comunicação entre pontos de extremidade é realizada através da conexão entre os grandes centros de dados (*data centers*), que são capazes de prover, sob demanda, serviços fim-a-fim adaptados a cada aplicação. Neste cenário multi-inquilino e multi-domínio, é imprescindível identificar a ocorrência de falhas e responsabilizar agentes maliciosos, que podem comprometer o bom comportamento e qualidade de serviço de milhares de usuários de um serviço simultaneamente. Portanto, uma solução de segurança baseada em corrente de blocos que garante confiança e imutabilidade dos registros em ambientes distribuídos é apropriada para solucionar estes desafios.

Com objetivo de apresentar um estudo de caso de corrente de blocos para garantir segurança em um ambiente de virtualização de redes, este capítulo apresentou as tecnologias de virtualização de redes de correntes de blocos. Em relação a corrente de blocos o capítulo apresentou a estrutura, propriedades e características de algumas correntes de blocos e, além disso, descreveu as categorias de correntes de blocos, comparando algoritmos de consenso e modelos de falha. Focou-se em mapear as principais características de correntes de blocos para atender às necessidades de segurança de ambientes de virtualização de funções de rede.

Também foi apresentado um estudo de caso de fatiamento de redes usando corrente de blocos. A ideia básica é propor correntes de blocos específicas e sob medida para cada cadeia de funções de rede que proveem serviços de rede. Assim, cada proposta atende a um requisito do ambiente virtualizado, como auditabilidade, autenticidade e rastreabilidade. No protótipo apresentado a decisão pelo protocolo de consenso proposto considerou a baixa quantidade e alta disponibilidade de nós identificados.

1.6.1. Escalabilidade e Vazão de Transações

Todos os protocolos de consenso e tipos de corrente de blocos apresentados possuem limitações em termos de taxa de transferência, latência de transação ou quantidade de nós [Popov 2016]. Os sistemas baseados em correntes de blocos ainda são incapazes de atingir o desempenho dos atuais sistemas centralizados de transferência de ativos. Enquanto as plataformas do PayPal e Visa processam aproximadamente 2000 transações por segundo em média e até 56.000 transações em pico [Visa 2019, PayPal 2019] com tempos de resposta da ordem de segundos [Wiki 2019], o Ethereum processa um máximo de 20 transações por segundo e o Bitcoin atinge uma vazão de apenas 7 transações por segundo. Ainda, o processo de mineração da prova de trabalho no Bitcoin gera gastos insustentáveis de energia sem retorno proporcional, atingindo em média 50 TW consumidos por ano [Digiconomist 2019]. Algumas correntes de blocos permissionadas atingem taxas da ordem de milhares de transações por segundo, porém reduzem o número de participantes possíveis [Schwartz et al. 2014, Buchman 2016, Kiayias et al. 2017].

Outro desafio de correntes de blocos é a eficiência no armazenamento e na busca de transações, pois, para prover segurança descentralizada, é necessário que todos os participantes do consenso processem todas as transações da rede e armazenem todo o histórico de transações. A verificação de transações é fundamental para garantir segurança em um ambiente sem confiança entre os pares como o de corrente de blocos. No entanto, verificar a existência ou corretude de uma transação pode envolver uma busca custosa em uma lista crescente de todas as transações presentes em todos os blocos da rede. Além disso, a quantidade de dados armazenados em cada participante da corrente de blocos cresce constantemente, uma vez que nenhuma transação ou bloco jamais é descartado. As principais implementações de correntes de blocos implementam estruturas auxiliares chamadas de árvores de Merkle para contornar o problema[17]. O trilema entre descentralização, segurança e escalabilidade dificulta que os sistemas baseados em correntes de blocos atinjam o desempenho de sistemas atuais e que atendam às necessidades do futuro.

1.6.2. Atividades em Organismos de Normalização

A aplicabilidade de da tecnologia de livro razão distribuído tem se demonstrado enorme. Assim, é essencial o pronto estabelecimento de normas internacionais para prover diretrizes relacionadas à uma nova tecnologia que propiciem um maior envolvimento e mais investimentos das indústrias. Em relação aos tópicos abordados neste capítulo, diferentes organismos de normalização criaram grupos de trabalho com objetivos variados.

A organização internacional de normalização (*International Organization for Standartization* - ISO) criou um comitê técnico que visa normalizar a tecnologia de corrente de blocos e a tecnologia de livro-razão distribuído [ISO/TC 307 2016]. O comitê possui projetos para formalizar os riscos de segurança, ameaças e vulnerabilidades da tecnologia, além de normalizar a arquitetura de referência, taxonomia, contratos inteligentes e a proteção de privacidade e de informações pessoais. A *Internet Research Task Force* (IRTF) criou o grupo de pesquisa da infraestrutura descentralizada da Internet (*Decentralized Internet Infrastructure Research Group* - DINRG) que estuda os serviços de infraestrutura beneficiados pela descentralização [IRTF 2017]. A *World Wide Web Consortium* (W3C) criou o grupo da comunidade de corrente de blocos (*Blockchain Community Group*) que tem como objetivo normalizar os formatos das mensagens em sistemas baseados em corrente de blocos [W3C 2016]. A comunidade usa o formato de mensagem definido na ISO 20022 como base para as normas. O grupo também busca definir diretrizes para o uso de armazenamento. A união internacional de telecomunicações (*International Telecommunications Union* - ITU), através do grupo de foco na aplicação da tecnologia de livro-razão distribuído (*Focus Group on Applications of Distributed Ledger Technology* - ITU-T FG DLT), pretende normalizar os serviços interoperáveis baseados na tecnologia de livro-razão distribuído [ITU-T FG DLT 2017]. O grupo de trabalho de corrente de blocos da Europa (*Europe Blockchain Working Group*) da associação internacional de segurança para comunicação institucional do comércio (*International Securities Association Trade Communication* - ISITC) discute a adoção da tecnologia de livro-razão distribuído pela indústria de serviços financeiros [Radford 2016].

[17] A estrutura e fundamentos de árvores de Merkle são apresentados no Apêndice B.

1.6.3. Papel da Corrente de Blocos em uma Economia Compartilhada

A economia do compartilhamento, ou compartilhada, é um modo de consumo onde bens e serviços não são de posse de um único usuário. Estes bens e serviços são temporariamente disponibilizados por outros indivíduos (terceiros), que recebem um incentivo financeiro para oferecer o serviço, através de uma plataforma que atua como intermediário entre o provedor do serviço e o consumidor. Os principais exemplos de empresas que utilizam este modelo são a Uber e a Airbnb, com outros exemplos incluindo: Cohealo, BlaBlaCar, JustPark, Skillshare, RelayRides e Landshare. Estas empresas conseguem seu lucro tomando uma parte dos ganhos dos usuários que provêm o serviço, ao fornecer uma plataforma capaz de conectar pessoas com interesses coincidentes.

A tecnologia de corrente de blocos permite prover confiança entre o provedor de serviço e o usuário do serviço sem necessidade de uma empresa intermediária. As empresas centralizadoras são assim eliminadas. [Hawlitschek et al. 2018] Além disso, contratos inteligentes permitem definir e impor regras para um acordo entre duas partes, possibilitando a fácil responsabilização de qualquer partícipe caso ocorra uma violação desse contrato. Plataformas de economia compartilhada implementadas com corrente de blocos são desse modo administradas coletivamente por todos seus usuários. Como é do interesse dos membros da plataforma que esta continue funcionando, há um incentivo para que os participantes atuem com honestidade.

Um exemplo de aplicação da corrente de blocos na economia compartilhada é a plataforma descentralizada de carona solidária Arcade City, que propõe um serviço similar ao Uber. Todo o processamento necessário para tratar do compartilhamento de caronas e outras funções é executado pela corrente de blocos. Como não existem intermediários, os custos das caronas podem ser reduzidos significativamente. Segurança é garantida usando um sistema de classificação no próprio aplicativo, garantindo que avaliações feitas por usuários encontrem-se sempre disponíveis e inalteradas graças a características da corrente de blocos.

1.6.4. Revogação de Chaves em Correntes de Blocos

Um dos principais desafios de sistemas baseados em correntes de blocos é a perda de chaves privadas, pois todas as transferências de ativos na rede são realizadas através de transações assinadas. A ausência de uma autoridade central confiável que armazene as identidades dos participantes da rede dificulta a revogação de chaves perdidas ou roubadas e consiste em um risco constante de perda de ativos. Em especial, é impossível revogar uma chave perdida em correntes de blocos públicas, nas quais o único identificador de um usuário é um endereço baseado na sua chave pública correspondente. Estima-se que mais de 30% dos *bitcoins* minerados estão inutilizados devido à perda de chaves, totalizando uma perda média de mais de 1000 *bitcoins* ou 8 milhões de dólares por dia [Chainalysis 2019].

Gerenciadores de chaves de correntes de blocos públicas implementam mecanismos de múltiplas assinaturas (*multisignature* ou *multisig*) para prevenir a perda de ativos decorrente da perda de uma chave privada. O mecanismo *multisignature* cria 3 pares de chaves assimétricas e exige a assinatura de pelo menos duas para emitir uma transação. As chaves podem ser armazenadas em locais diferentes e, possivelmente, controladas por

entidades diferentes. Caso uma chave seja perdida, é possível assinar com as duas restantes. Nenhuma chave pode emitir uma transação individualmente. O mecanismo permite tolerar perdas de até uma chave ou repartir a posse de um ativo, pois, caso as chaves pertençam a entidades diferentes, pelo menos duas entidades devem concordar para emitir uma transação. Caso uma chave seja perdida no mecanismo de assinaturas múltiplas, basta criar um novo endereço e emitir uma transação assinada transferindo o ativo com as duas chaves restantes. Sistemas baseados em correntes de blocos permissionadas podem implementar mecanismos baseados em identidades pessoais para criar listas locais de revogação de chaves que devem ser alteradas através de consenso [Androulaki et al. 2018].

1.6.5. Corrente de Blocos para Cidades Inteligentes

O uso de corrente de blocos em sistemas de Cidades Inteligentes possui restrições de segurança e privacidade que ainda são desafios [Khatoun and Zeadally 2016]. As correntes de dados públicas, como o Bitcoin, efetuam transações anônimas identificadas por chaves públicas. No entanto, a anonimidade não é absoluta, pois todas as transações são visíveis para todos os participantes da corrente de blocos e, assim, as atividades do usuário podem ser rastreadas. Combinando as informações dessas transações com alguns dados externos permite revelar a identidade real do usuário. Uma vez que a identidade do usuário é descoberta, todas suas ações serão rastreadas e dados confidencias, como padrões de compra e venda, serão vazados. Portanto, estes sistemas não garantem a privacidade dos dados dos usuários o que viola um dos princípios da segurança da informação [Talari et al. 2017].

1.6.6. O Uso de Corrente de Blocos na Área de Saúde

A alta produção e transferência de dados na área da saúde podem ser beneficiadas pelo uso de corrente de blocos. Sistemas baseados corrente de blocos permite que diferentes serviços de saúde possam compartilhar e armazenar dados e registros médicos em uma rede permissionada [Azaria et al. 2016]. Enquanto o uso de contratos inteligentes permite que pacientes controlem o acesso e a transferência de seus dados privados, a cópia da corrente de dados em diferentes locais e a auditabilidade atendem à demanda de acesso de diferentes interessados ao mesmo documento.

A corrente de blocos também possui aplicações na rastreabilidade de dados na área da saúde. Sistemas baseados em corrente de blocos permite o estabelecimento de uma rede privada e segura entre serviços de saúde para armazenar prescrições de remédios [Zhang et al. 2018]. As mudanças no estado ou posição do medicamento, assim como a transferência de propriedade são registradas no livro-razão distribuído, formando um registro histórico de dados do remédio desde a origem. A auditabilidade permite o acesso das entidades credenciadas às informações, facilitando a identificação de medicamentos falsos ou desviados.

1.6.7. Considerações e Perspectivas

A tecnologia de corrente de blocos é muito recente, pois tem apenas dez anos. O fato desta tecnologia prover uma camada de confiança distribuída faz com que ela seja considerada disruptiva e a tecnologia mais importante depois da Internet. As criptomo-

edas já são um sucesso e existem predições que afirmam que todo o dinheiro do mundo estará em uma corrente de blocos nos próximos dez anos. Existe uma série de aplicações de sucesso com contratos inteligentes e garantia de proveniência em diferentes áreas. Assiste-se hoje uma colaboração das indústrias sem precedentes mesmo entre empresas que eram concorrentes. O interesse na tecnologia de corrente de blocos cresce exponencialmente e o profissionais especialistas desta área estão muito requisitados e estão entre os mais bem pagos. Muitos desafios ainda persistem e muitos avanços são esperados para os próximos anos. Com certeza é uma área com enormes perspectivas e oportunidades.

A. Criptografia Assimétrica, Assinatura e Função *Hash*

Este apêndice apresenta alguns conceitos básicos de criptografia que são usados pelas correntes de blocos.

Criptografia Assimétrica

A criptografia assimétrica consiste em um sistema criptográfico baseado em um par de chaves assimétricas: uma chave pública e uma chave privada. Enquanto a chave pública pode ser amplamente disseminada, a chave privada deve ser mantida em segredo. Assim, a chave pública é usada para criptografar uma mensagem que apenas quem possui a chave privada, par desta chave pública, é capaz de descriptografar. A criptografia RSA (Rivest-Shamir-Adleman) e a criptografia de curvas elípticas são as principais tecnologias de chaves assimétricas. As duas tecnologias podem prover o mesmo nível de segurança, mas as curvas elípticas requerem chaves de menor tamanho.

Assinatura Digital

O mecanismo de assinatura digital deve prover as propriedades básicas de autenticidade, não repúdio e integridade. A propriedade de autenticidade deve permitir a confirmação da autenticidade de uma mensagem, ou seja, que somente o signatário deve ser capaz de gerar sua assinatura digital para aquela mensagem. O não repúdio deve garantir que o signatário de uma mensagem assinada digitalmente não possa negar a autoria da assinatura. Por fim, é necessário a garantia de integridade: para sustentar as características anteriores de autenticidade e de não repúdio, a mensagem não pode ser adulterada.

Uma mensagem criptografada por uma chave privada só pode ser descriptografada pelo seu par de chave pública. Esta operação é conhecida como assinatura, pois identifica inequivocamente a pessoa que possui a chave privada, que é secreta e apenas seu proprietário possui.

A utilização de curvas elípticas em criptografia foi sugerida por Neal Koblitz e Victor S. Miller em 1985. Uma curva elíptica é o locus dos pontos do plano cujas coordenadas satisfazem a equação cúbica $y^2 = x^3 + ax + b$ junto com um ponto na infinidade O. A Figura 1.12 ilustra a curva elíptica "secp256k1", usada na criptografia assimétrica do Bitcoin e definida nas normas para criptografia eficiente (*Standards for Efficient.00Cryptography* - SEC)[18].

O algoritmo de assinatura digital por curvas elípticas (*Elliptic Curve Digital Sig-*

[18]Certicom Research, http://www.secg.org/sec2-v2.pdf

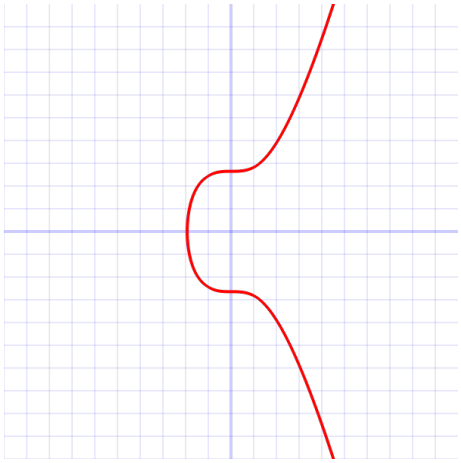


Figura 1.12. Curva Elíptica secp256k1 usada na moeda eletrônica Bitcoin.

nature Algorithm - ECDSA é uma variante do tradicional algoritmo de assinatura digital (*Digital Signature Algorithm* - DSA). O algoritmo baseado em curvas elípticas implementa uma modificação da norma de assinatura digital (*Digital Signature Standard* - DSS) que executa operações sobre pontos de curvas elípticas, ao invés das operações de exponenciação usadas no DSS. A maior vantagem do ECDSA sobre o DSA é que o ECDSA requer tamanhos bem menores de chave para propiciar a mesma segurança[19].

Função *Hash*

Uma função *hash* ou função resumo é uma função que mapeia dados de comprimento variável em dados de comprimento fixo. Uma função *hash* criptográfica é uma transformação matemática que, a partir de uma mensagem de comprimento arbitrário ou uma sequência de bits de tamanho arbitrário, obtém uma sequência curta e com tamanho fixo de bits. A função *hash*, descrita por $H(m) = h$, onde m é a mensagem em claro, $H()$ é a função *hash* e h é o valor *hash* resultante, deve ter as seguintes propriedades:

- a função *hash*, $H(m)$, pode ser aplicada a uma mensagem, m, de qualquer tamanho;
- a função *hash*, $H(m)$, gera uma saída h de tamanho fixo (por exemplo, o tamanho é 256 bits se $H(m)$ for o algoritmo SHA256);
- a função *hash* é simples, ou seja, é computacionalmente eficiente calcular $H(m)$ para qualquer m (a computação de $H(m)$ utiliza operações lógicas e evita as multiplicações e exponenciações usadas em criptografia assimétrica);
- a função *hash* é unidirecional ou não-invertível, isto é, para um h qualquer é computacionalmente impossível achar m tal que $H(m) = h$;
- a função *hash* é livre de colisões, ou seja, para uma mensagem, m, é computacionalmente impossível achar uma mensagem diferente, m', tal que $H(m) = H(m')$. Deve

[19]Uma chave de criptografia de curvas elípticas (*Eliptic Curve Criptography* -ECC) de 163 bits corresponde à mesma garantia de segurança de uma chave RSA (*Rivest-Shamir-Adleman*) de 3.072 bits e uma chave de criptografia simétrica AES (*Advanced Encription Standard*) de 128 bits.

ser ressaltado que embora possam existir muitas mensagens que geram o mesmo *hash*, a função *hash* criptográfica deve ser projetada para dificultar ao máximo a colisão de *hashes*;

A aplicação principal da função *hash* é a garantia de integridade de uma mensagem. Em geral, o emissor de uma mensagem calcula o seu *hash* e o insere no final da mensagem. O receptor recalcula o *hash* da mensagem recebida e compara o resultado obtido com o *hash* recebido. Se os *hashes* forem diferentes, há garantia de que a mensagem recebida é diferente da mensagem enviada. Se os *hashes* forem iguais, há uma probabilidade muito alta da mensagem recebida ser igual a mensagem enviada. Assim, qualquer alteração na mensagem pode ser detectada pelo recálculo e comparação do *hash*, o que garante a integridade da mensagem.

Existem diferentes funções *hash*, mas o Bitcoin e maioria das moedas criptográficas usam a função *hash* SHA256, que resulta em 256 bits. O algoritmo seguro de *hash* (*Secure Hash Algorithm* - SHA) foi desenvolvido pela Agência de Segurança Nacional (*National Security Agency* - NSA) dos EUA.

Assinatura da Mensagem e Assinatura do *Hash* da Mensagem

Uma mensagem criptografada por uma chave privada só pode ser descriptografada pela chave pública correspondente à chave privada que criptografou a mensagem. Esta operação garante a autenticidade do emissor, pois só ele possui a chave privada que foi usada na criptografia, e qualquer um que possua sua chave pública correspondente pode verificar a mensagem. Porém, a mensagem pode ter sido adulterada no caminho. Por outro lado, se o emissor computa o *hash* da mensagem, criptografa o *hash* com sua chave privada e envia o *hash* criptografado para o destinatário junto da mensagem, esta operação garante tanto a integridade da mensagem quanto autenticidade do emissor. A mensagem vai em claro e portanto não garante o sigilo, mas o destinatário garante ao mesmo tempo que a mensagem chegou íntegra e que o emissor é autêntico. Para isso, basta recalcular o *hash* a partir da mensagem, descriptografar o resultado com a chave pública do emissor e compará-lo com o *hash* assinado contido na mensagem.

B. Árvores de Merkle e Verificação Simples de Pagamento

As árvores de Merkle são a principal estrutura auxiliar utilizada em correntes de blocos para verificar a integridade e não-repúdio de uma transação de maneira eficiente [Nakamoto 2008]. Nomeadas em homenagem a Ralph Merkle, que patenteou o conceito em 1979, as árvores de Merkle são estruturas de dados em forma de árvore na qual cada nó não-folha, denominado de "nó-galho,"é o resultado de um *hash* de seus respectivos nós filhos. Cada nó-folha da árvore é o resultado de um *hash* de um conjunto de dados que, no caso da corrente de blocos, representam as transações da rede. A Figura 1.13 ilustra a estrutura de uma corrente de blocos que utiliza uma árvore de Merkle binária para armazenar transações. A estrutura da árvore pode ser construída calculando o *hash* de cada transação T para criar o nível mais baixo da árvore, e, posteriormente, utilizando os *hashes* de cada nível para construir o próximo nível, até chegar à raiz da árvore. A complexidade de construção de uma árvore de Merkle de um bloco é $O(n.log_2(n)$ onde n é o número de transações contidas no bloco. A árvore binária é o tipo de árvore de

Merkle mais utilizado em correntes de blocos, porém algumas implementações utilizam tipos diferentes de árvore [Wood 2014].

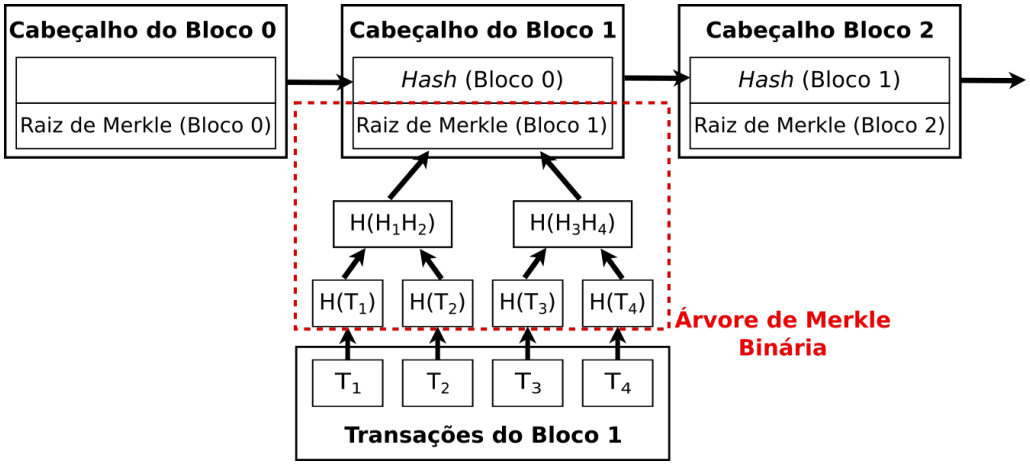


Figura 1.13. Estrutura de uma árvore de Merkle binária utilizada em correntes de blocos. O uso de árvores de Merkle permite armazenar apenas a raiz da árvore sem prejuízo à verificação das transações.

O uso de árvores Merkle permite obter uma prova de Merkle (*Merkle proof*), que verifica a presença e a corretude de uma transação em um bloco de maneira eficiente. Para verificar uma transação, basta fornecer os *hashes* necessários para reconstruir o caminho da árvore correspondente à transação verificada. A reconstrução de um caminho e, logo, a verificação de uma transação, tem complexidade $O(n)$. Assim, é possível verificar transações rapidamente mesmo em meio a milhares de transações, e não é necessário armazenar o bloco inteiro para verificar uma transação. Além disso, o uso de *hashes* diminui o tráfego na rede, pois não é necessário transferir a transação ou bloco completos. Isto permite criar "nós leves"e um mecanismo simplificado de verificação, pois, para verificar uma transação, basta armazenar o cabeçalho do bloco e solicitar a nós que possuem todas as transações os *hashes* do caminho até a transação. As provas de Merkle são a base do mecanismo de verificação simples de pagamento (*Simple Payment Verification* - SPV) proposto por Nakamoto e utilizado na maioria das implementações de corrente de blocos atuais [Nakamoto 2008, Wood 2014].

Referências

[Alchieri et al. 2018] Alchieri, E. A. P., Bessani, A., Greve, F., and d. S. Fraga, J. (2018). Knowledge Connectivity Requirements for Solving Byzantine Consensus with Unknown Participants. *IEEE Transactions on Dependable and Secure Computing*, 15(2):246–259.

[Ali et al. 2016] Ali, M., Nelson, J. C., Shea, R., and Freedman, M. J. (2016). Blockstack: a Global Naming and Storage System Secured by Blockchains. In *USENIX Annual Technical Conference*, pages 181–194.

[Alvarenga 2018] Alvarenga, I. D. (2018). Securing Configuration, Management and Migration of Virtual Network Functions using Blockchain. Master Science Thesis, Universidade Federal do Rio de Janeiro, Brasil.

[Alvarenga et al. 2018] Alvarenga, I. D., Rebello, G. A. F., and Duarte, O. C. M. B. (2018). Securing management, configuration, and migration of virtual network functions using blockchain. In *IEEE/IFIP NOMS*.

[Androulaki et al. 2018] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S. W., and Yellick, J. (2018). Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 30:1–30:15, New York, NY, USA. ACM.

[Angelis et al. 2018] Angelis, S. D., Aniello, L., Baldoni, R., Lombardi, F., Margheri, A., and Sassone, V. (2018). PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain. In *Italian Conference on Cyber Security (06/02/18)*.

[Arnott et al. 2016] Arnott, M., Owen, P., Buckland, E., and Ranken, M. (2016). IoT global forecast analysis 2015-2025. Technical report, Machina Research. `https://machinaresearch.com/login/?next=/report/iot-global-forecast-analysis-2015-25/`. Acessado em 09 de abril de 2019.

[Azaria et al. 2016] Azaria, A., Ekblaw, A., Vieira, T., and Lippman, A. (2016). Medrec: Using blockchain for medical data access and permission management. In *Open and Big Data (OBD), International Conference on*, pages 25–30. IEEE.

[Back 2002] Back, A. (2002). Hashcash - a denial of service counter-measure. `http://www.hashcash.org/papers/hashcash.pdf`. Acessado em 9 de abril de 2019.

[Bessani et al. 2014] Bessani, A., Sousa, J., and Alchieri, E. E. P. (2014). State Machine Replication for the Masses with BFT-SMART. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362.

[Bhamare et al. 2016] Bhamare, D., Jain, R., Samaka, M., and Erbad, A. (2016). A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155.

[Bouras et al. 2017] Bouras, C., Kollia, A., and Papazois, A. (2017). SDN & NFV in 5G: Advancements and challenges. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 107–111.

[Bozic et al. 2017] Bozic, N., Pujolle, G., and Secci, S. (2017). Securing Virtual Machine Orchestration with Blockchains. In *2017 1st Cyber Security in Networking Conference*.

[Brewer 2000] Brewer, E. A. (2000). Towards Robust Distributed Systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, New York, NY, USA. ACM.

[Buchman 2016] Buchman, E. (2016). *Tendermint: byzantine fault tolerance in the age of blockchains*. PhD thesis, The University of Guelph.

[Cachin and Vukolić 2017] Cachin, C. and Vukolić, M. (2017). Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*.

[Chainalysis 2019] Chainalysis (2019). Crypto Crime Report: Decoding increasingly sophisticated hacks, darknet markets, and scams. Technical report, Chainalysis Inc.

[Chaum 1983] Chaum, D. (1983). Blind signatures for untraceable payments. In *Advances in cryptology*, pages 199–203. Springer, Hong-Ning.

[Chicarino et al. 2017] Chicarino, V. R. L., Jesus, E. F., de Albuquerque, C. V. N., and de A. Rocha, A. A. (2017). Uso de Blockchain para Privacidade e Segurança em Internet das Coisas. In *Minicursos do SBSeg'2017*, pages 149–200.

[Christidis and Devetsikiotis 2016] Christidis, K. and Devetsikiotis, M. (2016). Blockchains and smart contracts for the Internet of Things. *IEEE Access*, 4:2292–2303.

[Dai 1998] Dai, W. (1998). B-money. `http://www.weidai.com/bmoney.txt`. Acessado em 9 de abril de 2019.

[Decker et al. 2016] Decker, C., Seidel, J., and Wattenhofer, R. (2016). Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, page 13. ACM.

[Digiconomist 2019] Digiconomist (2019). Bitcoin Energy Consumption Index. `https://digiconomist.net/bitcoin-energy-consumption`. Acessado em 9 de abril de 2019.

[Dinh et al. 2017] Dinh, T. T. A., Wang, J., Chen, G., Liu, R., Ooi, B. C., and Tan, K.-L. (2017). Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1085–1100. ACM.

[Dolev 1982] Dolev, D. (1982). The Byzantine generals strike again. *Journal of Algorithms*, 3(1):14 – 30.

[Dolev and Yao 1983] Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208.

[Douceur 2002] Douceur, J. R. (2002). The Sybil Attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260, London, UK, UK. Springer-Verlag.

[Duan et al. 2018] Duan, S., Reiter, M. K., and Zhang, H. (2018). BEAT: Asynchronous BFT made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2028–2041. ACM.

[Dwork et al. 1988] Dwork, C., Lynch, N., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323.

[ETSI 2014] ETSI (2014). ETSI GS NFV-MAN 001: Network Functions Virtualisation; Management and Orchestration.

[Eyal et al. 2016] Eyal, I., Gencer, A. E., Sirer, E. G., and Van Renesse, R. (2016). Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 45–59.

[Eyal and Sirer 2018] Eyal, I. and Sirer, E. G. (2018). Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102.

[Finney 2005] Finney, H. (2005). RPOW: Reusable proofs of work. `http://nakamotoinstitute.org/finney/rpow/theory.html`. Acessado em 9 de abril de 2019.

[Fischer et al. 1985] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382.

[Frantz and Nowostawski 2016] Frantz, C. K. and Nowostawski, M. (2016). From institutions to code: Towards automated generation of smart contracts. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 210–215. IEEE.

[Greve et al. 2018] Greve, F., Sampaio, L., Abijaude, J., Coutinho, A. A. R., Brito, I. V. S., and Queiroz, S. (2018). Blockchain e a Revolução do Consenso sob Demanda. In *Minicursos do SBRC'2018*, volume 36.

[Halpern and Pignataro 2015] Halpern, J. and Pignataro, C. (2015). Service Function Chaining (SFC) architecture. RFC 7665, RFC Editor. `http://www.rfc-editor.org/rfc/rfc7665.txt`. Acessado em 9 de abril de 2019.

[Han et al. 2015] Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.

[Han et al. 2018] Han, R., Gramoli, V., and Xu, X. (2018). Evaluating blockchains for iot. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5.

[Hawlitschek et al. 2018] Hawlitschek, F., Notheisen, B., and Teubner, T. (2018). The limits of trust-free systems: A literature review on blockchain technology and trust in the sharing economy. *Electronic commerce research and applications*, 29:50–63.

[Huh et al. 2017] Huh, S., Cho, S., and Kim, S. (2017). Managing IoT devices using blockchain platform. In *Advanced Communication Technology (ICACT), 19th International Conference on*, pages 464–467. IEEE.

[IRTF 2017] IRTF (2017). Decentralized Internet Infrastructure Research Group. `https://trac.ietf.org/trac/irtf/wiki/blockchain-federation`. Acessado em 9 abril de 2019.

[ISO/TC 307 2016] ISO/TC 307 (2016). Blockchain and distributed ledger technologies. `https://www.iso.org/committee/6266604.html`. Acessado em 9 de abril de 2019.

[ITU-T FG DLT 2017] ITU-T FG DLT (2017). Focus Group on Application of Distributed Ledger Technology. `https://itu.int/en/ITU-T/focusgroups/dlt/Pages/default.aspx`. Acessado em 9 de abril de 2019.

[Khatoun and Zeadally 2016] Khatoun, R. and Zeadally, S. (2016). Smart Cities: Concepts, Architectures, Research Opportunities. *Commun. ACM*, 59(8):46–57.

[Kiayias et al. 2017] Kiayias, A., Russell, A., David, B., and Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer.

[Lamport 1998] Lamport, L. (1998). The Part-time Parliament. *ACM Transactions on Computer Systems*, 16(2):133–169.

[Lamport et al. 1982] Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401.

[Lee et al. 2016] Lee, K., James, J. I., Ejeta, T. G., and Kim, H. J. (2016). Electronic voting service using block-chain. *Journal of Digital Forensics, Security and Law*, 11(2):8.

[Li and Chen 2015] Li, Y. and Chen, M. (2015). Software-Defined Network Function Virtualization: A Survey. *IEEE Access*, 3:2542–2553.

[Mattos et al. 2018] Mattos, D. M. F. et al. (2018). Blockchain para Segurança em Redes Elétricas Inteligentes: Aplicações, Tendências e Desafios. In *Minicursos do SBSeg'2018*, pages 140–194.

[Medhat et al. 2017] Medhat, A. M., Taleb, T., Elmangoush, A., Carella, G. A., Covaci, S., and Magedanz, T. (2017). Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges. *IEEE Communications Magazine*, 55(2):216–223.

[Mello et al. 2017] Mello, A. M., Marino, F. C. H., and Santos, R. R. (2017). Segurança de Aplicações Blockchain Além das Criptomoedas. In *Minicursos do SBSeg'2017*, pages 99–148.

[Mijumbi et al. 2016] Mijumbi, R., Serrat, J., Gorricho, J., Bouten, N., De Turck, F., and Boutaba, R. (2016). Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262.

[Miller et al. 2016] Miller, A., Xia, Y., Croman, K., Shi, E., and Song, D. (2016). The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 31–42, New York, NY, USA. ACM.

[Mingxiao et al. 2017] Mingxiao, D., Xiaofeng, M., Zhe, Z., Xiangwei, W., and Qijun, C. (2017). A review on consensus algorithm of blockchain. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2567–2572. IEEE.

[Nakamoto 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. `http://bitcoin.org/bitcoin.pdf`. Acessado em 9 de abril de 2019.

[Olson et al. 2018] Olson, K., Bowman, M., Mitchell, J., Amundson, S., Middleton, D., and Montgomery, C. (2018). Sawtooth: An Introduction. *The Linux Foundation, Jan*.

[Ongaro and Ousterhout 2014] Ongaro, D. and Ousterhout, J. (2014). In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pages 305–320, Berkeley, CA, USA. USENIX Association.

[Pattaranantakul et al. 2016] Pattaranantakul, M., He, R., Meddahi, A., and Zhang, Z. (2016). SecMANO: Towards Network Functions Virtualization (NFV) Based Security MANagement and Orchestration. In *IEEE Trustcom/BigDataSE/ISPA*, pages 598–605.

[Paul et al. 2014] Paul, G., Sarkar, P., and Mukherjee, S. (2014). Towards a More Democratic Mining in Bitcoins. In *International Conference on Information Systems Security*, pages 185–203. Springer International Publishing.

[PayPal 2019] PayPal (2019). Paypal Holdings, Inc. (PYPL) SEC Filing 10-K Annual report for the fiscal year ending Monday, December 31, 2018. `https://filings.last10k.com/sec-filings/1633917/000163391719000043/pypl201810-k.htm.pdf`. Acessado em 09 de Abril de 2019.

[Popov 2016] Popov, S. (2016). The Tangle. `https://iota.org/IOTAWhitepaper.pdf`. Acessado em 9 de abril de 2019.

[Radford 2016] Radford, D. (2016). Europe Blockchain Working Group. Standard, The International Securities Association for Institutional Trade Communication. `https://isitc-europe.com/isitc-europe-blockchain-working-group`. Acessado em 9 de abril de 2019.

[Raman 1998] Raman, L. (1998). OSI systems and network management. *IEEE Communications Magazine*, 36(3):46–53.

[Rebello 2019] Rebello, G. A. F. (2019). Encadeamento Seguro de Funções Virtuais de Rede Baseado em Correntes de Blocos. Projeto Final de Graduação, Universidade Federal do Rio de Janeiro, Brasil.

[Rebello et al. 2018] Rebello, G. A. F., Alvarenga, I. D., Sanz, I. J., and Duarte, O. C. M. B. (2018). SINFONIA: Gerenciamento Seguro de Funções Virtualizadas de Rede através de Corrente de Blocos. In *I Workshop em Blockchain: Teoria, Tecnologias e Aplicações (WBlockchain - SBRC 2018)*, Campos do Jordão, SP, Brasil. SBC.

[Rebello et al. 2019a] Rebello, G. A. F., Alvarenga, I. D., Sanz, I. J., and Duarte, O. C. M. B. (2019a). BSec-NFVO: A Blockchain-based Security for Network Function Virtualization Orchestration. In *IEEE International Conference on Communications (ICC)*. A ser publicado.

[Rebello et al. 2019b] Rebello, G. A. F., Camilo, G. F., Silva, L. G. C., Guimarães, L. C. B., de Souza, L. A. C., Alvarenga, I. D., and Duarte, O. C. M. B. (2019b). Provendo uma infraestrutura de software fatiada, isolada e segura de funções virtuais através da tecnologia de corrente de blocos. Technical report, Grupo de Teleinformática e Automação (GTA/COPPE/UFRJ).

[Santos and Schiper 2012] Santos, N. and Schiper, A. (2012). Tuning Paxos for High-throughput with Batching and Pipelining. In *Proceedings of the 13th International Conference on Distributed Computing and Networking*, ICDCN'12, pages 153–167, Berlin, Heidelberg. Springer-Verlag.

[Schneider 1993] Schneider, F. B. (1993). chapter Replication Management Using the State-machine Approach. In Mullender, S., editor, *Distributed Systems (2Nd Ed.)*, pages 169–197. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

[Schwartz et al. 2014] Schwartz, D., Youngs, N., and Britto, A. (2014). The Ripple Protocol Consensus Algorithm. *Ripple Labs Inc White Paper*, 5.

[Sekar et al. 2012] Sekar, V., Egi, N., Ratnasamy, S., Reiter, M. K., and Shi, G. (2012). Design and Implementation of a Consolidated Middlebox Architecture. In *9th Symposium on Networked Systems Design and Implementation (NSDI)*, pages 323–336, San Jose, CA. USENIX.

[Smolensk 2018] Smolensk, M. (2018). Lightstreams White Paper. `https://s3.amazonaws.com/lightstreams/lightstreams_whitepaper.pdf.Acessadoem9deabrilde2019.`

[Sousa et al. 2018] Sousa, J., Bessani, A., and Vukolic, M. (2018). A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 51–58.

[Statista 2018] Statista (2018). Internet of Things - number of connected devices worldwide. Technical report, Statista. `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`. Acessado em 9 de abril de 2019.

[Storj Labs 2018] Storj Labs, I. (2018). Storj: A Decentralized Cloud Storage Network Framework. `https://storj.io/storjv3.pdf`. Acessado em 9 de abril de 2019.

[Talari et al. 2017] Talari, S., Shafie-khah, M., Siano, P., Loia, V., Tommasetti, A., and Catalão, J. P. S. (2017). A Review of Smart Cities Based on the Internet of Things Concept. *Energies*, 10(4).

[Tikhomirov 2018] Tikhomirov, S. (2018). Ethereum: State of Knowledge and Research Perspectives. In *Foundations and Practice of Security*, pages 206–221. Springer International Publishing.

[Tseng 2017] Tseng, L. (2017). Bitcoin's Consistency Property. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 219–220.

[Vasin 2014] Vasin, P. (2014). Blackcoin's Proof-of-Stake Protocol v2. *URL: https://blackcoin. co/blackcoin-pos-protocol-v2-whitepaper. pdf*, 71.

[Visa 2019] Visa (2019). About VisaNet. `https://usa.visa.com/about-visa/visanet.html`. Acessado em 9 de Abril de 2019.

[Vukolić 2016] Vukolić, M. (2016). *The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication*, pages 112–125. Springer International Publishing, Cham.

[W3C 2016] W3C (2016). Blockchain Community Group. `https://www.w3.org/community/blockchain`. Acessado em 9 de abril de 2019.

[Wiki 2019] Wiki, B. (2019). Bitcoin Scalability. `https://en.bitcoin.it/wiki/Scalability`. Acessado em 9 de abril de 2019.

[Wilkinson et al. 2014] Wilkinson, S., Boshevski, T., Brandoff, J., and Buterin, V. (2014). Storj: a Peer-to-Peer Cloud Storage Network. `https://storj.io/storj2014.pdf`. Acessado em 9 de abril de 2019.

[Wood 2014] Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. `http://gavwood.com/paper.pdf`. Acessado em 9 de abril de 2019.

[Wüst and Gervais 2018] Wüst, K. and Gervais, A. (2018). Do you Need a Blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54.

[Xu et al. 2017] Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., and Rimba, P. (2017). A Taxonomy of Blockchain-Based Systems for Architecture Design. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 243–252.

[Zhang et al. 2018] Zhang, P., Schmidt, D. C., White, J., and Lenz, G. (2018). Chapter One - Blockchain Technology Use Cases in Healthcare. In Raj, P. and Deka, G. C., editors, *Blockchain Technology: Platforms, Tools and Use Cases*, volume 111 of *Advances in Computers*, pages 1 – 41. Elsevier.

[Zheng et al. 2018] Zheng, Z., Xie, S., Dai, H.-N., Chen, X., and Wang, H. (2018). Blockchain Challenges and Opportunities: A Survey. *International Journal of Web and Grid Services*, 14(4):352–375.

Capítulo 2

Computação Urbana da Teoria à Prática: Fundamentos, Aplicações e Desafios

Diego O. Rodrigues (IC-UNICAMP), Frances A. Santos (IC-UNICAMP), Geraldo P. Rocha Filho (CiC-UnB), Ademar T. Akabane (IC-UNICAMP), Raquel Cabral (UFAL), Roger Immich (IC-UNICAMP), Wellington L. Junior (IC-UNICAMP), Felipe D. Cunha (PUC-Minas), Daniel L. Guidoni (UFSJ), Thiago H. Silva (UTFPR), Denis Rosário (UFPA), Eduardo Cerqueira (UFPA), Antonio A. F. Loureiro (UFMG) e Leandro A. Villas (IC-UNICAMP)

Abstract

The growing of cities has resulted in innumerable technical and managerial challenges for public administrators such as energy consumption, pollution, urban mobility and even supervision of private and public spaces in an appropriate way. Urban Computing emerges as a promising paradigm to solve such challenges, through the extraction of knowledge, from a large amount of heterogeneous data existing in urban space. Moreover, Urban Computing correlates urban sensing, data management, and analysis to provide services that have the potential to improve the quality of life of the citizens of large urban centers. Consider this context, this chapter aims to present the fundamentals of Urban Computing and the steps necessary to develop an application in this area. To achieve this goal, the following questions will be investigated, namely: (i) What are the main research problems of Urban Computing?; (ii) What are the technological challenges for the implementation of services in Urban Computing?; (iii) What are the main methodologies used for the development of services in Urban Computing?; and (iv) What are the representative applications in this field?

Resumo

A rápida urbanização das cidades vem resultando em inúmeros desafios técnicos e gerenciais para os gestores públicos tais como consumo de energia, poluição, mobilidade urbana e até mesmo gestão de espaços privados e públicos de forma apropriada. A Computação Urbana surge como um paradigma promissor para resolver tais desafios,

por meio da extração de informação, a partir de uma grande quantidade de dados heterogêneos existentes no espaço urbano. Mais ainda, a Computação Urbana correlaciona o sensoriamento urbano, o gerenciamento de dados e sua análise para fornecer serviços que têm o potencial de melhorar a qualidade da vida dos habitantes de grandes centros urbanos. Com isso em mente, este minicurso tem como objetivo apresentar os fundamentos da Computação Urbana e os passos necessários para desenvolver uma aplicação nessa área. Para alcançar tal objetivo, serão investigadas as seguintes questões, a saber: (i) Quais são os principais problemas de pesquisa da Computação Urbana?; (ii) Quais são os desafios tecnológicos existentes para a implantação de serviços na Computação Urbana?; (iii) Quais são as principais metodologias utilizadas para o desenvolvimento de serviços na Computação Urbana?; e (iv) Quais são as aplicações representativas neste domínio?

2.1. Introdução

Um relatório da ONU mostra que atualmente 54% da população mundial vive em áreas urbanas, e que até 2050 espera-se que esse número chegue a 66% [United Nations and Affairs 2018]. Dessa forma, essa rápida urbanização das cidades resulta em inúmeros desafios técnicos e gerenciais para os gestores públicos, tais como consumo de energia, poluição, mobilidade urbana e até mesmo gestão de espaços privados e públicos de forma apropriada [Zheng et al. 2014, Filho et al. 2018].

Nesse contexto, a Computação Urbana surge como um paradigma promissor para superar tais desafios típicos de centros urbanos, por meio da extração de informação, a partir de uma grande quantidade de dados heterogêneos existentes nas cidades [Zheng et al. 2014]. Este campo de estudo visa aplicar soluções computacionais para melhorar os problemas enfrentados em grandes cidades. Para isso, a Computação Urbana utiliza diferentes tecnologias de sensoriamento remoto, técnicas de gerenciamento de dados e modelos analíticos, além de métodos de visualização para criar soluções que melhoram a qualidade de vida da população e dos sistemas de gerenciamento dos serviços oferecidos pelas cidades. Uma questão importante é que a Computação Urbana pode auxiliar a entender a natureza dos fenômenos urbanos ou até mesmo prevê-los [Zheng et al. 2014, Silva et al. 2018].

De uma maneira geral, pode-se apresentar a Computação Urbana composta por três macro atividades [Zheng et al. 2014, Silva et al. 2018]: (i) gerenciamento dos dados urbanos; (ii) análise dos dados urbanos; e (iii) desenvolvimento de serviços e aplicações para cidades inteligentes, conforme exibido na Figura 2.1. O gerenciamento de dados urbanos consiste no processo de coleta de dados urbanos, no processamento desses dados e na modelagem dos mesmos. A análise dos dados urbanos é realizada por meio da implementação de algoritmos, bem como da interpretação de resultados. Outra atividade é o desenvolvimento de serviços e aplicações com o conhecimento obtido.

O minicurso tem como objetivo principal introduzir os fundamentos da Computação Urbana e apresentar as ferramentas e os maios necessários para desenvolver uma aplicação nessa área. Os conceitos englobam uma visão geral dos trabalhos existentes, os desafios e as perspectivas futuras nessa área em questão. A Computação Urbana é um tópico de pesquisa emergente, dessa forma existem problemas em aberto e desafios que não são triviais de serem resolvidos. Por causa disso, este minicurso também possui a finalidade

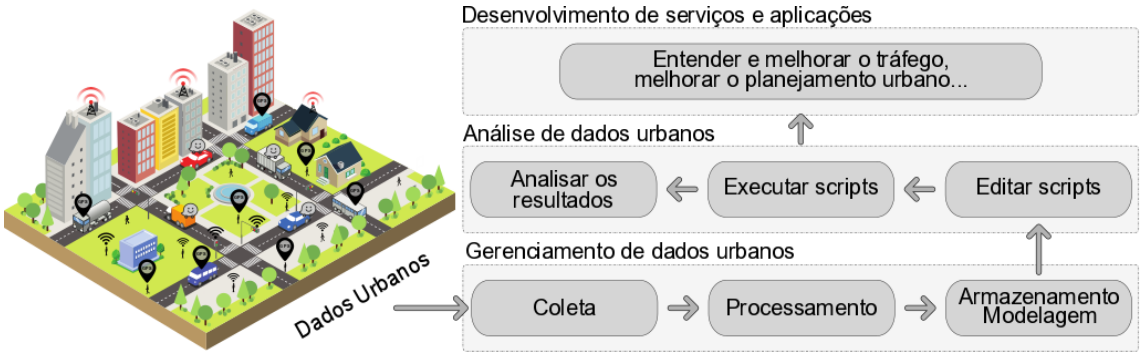


Figura 2.1. Visão geral do arcabouço da computação urbana. [Inspirada em [Silva et al. 2018]].

de investigar as seguintes questões, as quais são retificadas por [Zheng et al. 2014]: (i) Quais são os principais problemas de pesquisa da Computação Urbana?; (ii) Quais são os desafios tecnológicos existentes para a implantação de serviços na Computação Urbana?; (iii) Quais são as principais metodologias utilizadas para o desenvolvimento de serviços na computação urbana?; e (iv) Quais são as aplicações representativas neste domínio?

O restante deste minicurso está organizado da seguinte forma. A seção 2.2 apresenta os fundamentos teóricos e os desafios relacionados à Computação Urbana. A seção 2.3 apresenta as principais aplicações e serviços que podem ser desenvolvidas com base na Computação Urbana. A seção 2.4 apresenta diversos tópicos de pesquisas relacionados com a Computação Urbana, os desafios e as questões de pesquisas em aberto. A seção 2.5 mostra uma perspectiva prática do desenvolvimento de um serviço com base em Computação Urbana. E por fim a seção 2.6 traz as conclusões a respeito do tema.

2.2. Computação Urbana

A demanda por aplicações relacionadas a computação urbana está em expansão. De acordo com as Nações Unidas, 55% da população atualmente vivem em áreas urbanas, no entanto, existem projeções que apontam que este número deverá aumentar para 68% até 2050 [United Nations and Affairs 2018]. Sendo assim, existe uma grande carência de métodos para oferecer uma infraestrutura adequada (como transporte, habitação, água e energia) para as grandes cidades de hoje e também preparar novas cidades para receber cada vez mais habitantes.

Visando uma solução holística para o problema, a computação urbana faz uso de uma grande quantidade de fonte de dados, como de dispositivos da Internet das Coisas (IoT) [Xu et al. 2014, Filho et al. 2019]; dados de redes sociais baseadas em localização (LBSN) [Silva et al. 2018, Zheng 2011, Zheng 2012, Traynor and Curran 2012] e também dados estatísticos sobre cidades e sua população. Neste sentido, a computação em névoa oferece um diferencial importante em relação aos demais paradigmas, pois os seus elementos processadores estão dispersos no espaço urbano, provendo assim uma rápida resposta e também a possibilidade de aplicações relacionadas ao contexto de cada região, como pode ser visto em [Pereira Rocha Filho et al. 2018, Filho et al. 2013, Geraldo Filho et al. 2018]. Um exemplo pode ser a identificação em tempo real de problemas. Sistemas de vigilância urbana podem receber suporte da névoa, permitindo automatizar tarefas e realizar a tomada de decisões em tempo real conforme a localização e contexto

de cada região [Chen et al. 2016].

De uma forma mais abrangente, a computação urbana busca os aspectos que permeiam os fenômenos urbanos, bem como fornecer estimativas sobre o futuro das cidades. Por este motivo, dentro da ciência da computação esse é um paradigma interdisciplinar, pois faz interseção com áreas que vão desde a de redes de computadores, redes de sensores e redes de veículos, interagindo também com sistemas distribuídos, inteligência artificial e redes sociais [Silva et al. 2018].

Conforme mencionado anteriormente, uma definição mais específica da computação urbana pode ser alcançada através de três macro atividades, nomeadamente (i) gerenciamento dos dados urbanos; (ii) análise dos dados urbanos; e (iii) desenvolvimento de serviços e aplicações para cidades inteligentes. A primeira etapa (i) gerenciamento dos dados urbanos diz respeito a coleta e processamento das informações. Esta tem o objetivo de obter amostras de dados a partir de várias fontes de interesse de forma eficiente e contínua. A atividade de amostragem dos dados pode ser realizada a partir de fontes dinâmicas e heterogêneas, já que dados oriundos de apenas uma fonte pode não oferecer completude. Um dos desafios neste sentido é a criação de técnicas para reduzir os ruídos e erros durante a coleta de grandes fluxos de dados, tento em vista que a utilização de diferentes fontes pode resultar em dados conflitantes e/ou duplicados [Guo et al. 2014].

Existem hoje, diversas fontes de dados publicamente disponíveis na Internet. Estas vão desde fontes com dados estatísticos sobre as cidades até dados das redes de sensoriamento participativo. Apesar da possibilidade, de uma forma geral, ainda existem poucas cidades com projetos de compartilhamento de dados. Outra dificuldade nesta área é a diversidade dos formatos nos quais os dados estão sendo disponibilizados, pois não existe um padrão pré-definido para isto. Portanto, essa etapa também precisa se preocupar com processamento e formatação dos dados recolhidos. Além disto, buscando suprir a carência de dados em cidades que não tem programas de compartilhamento, é possível utilizar dados de redes sociais para obter informações similares aos dados estatísticos oficiais [Silva et al. 2013, Silva et al. 2014].

Outra possibilidade de obtenção de dados é através do sensoriamento participativo. Essas informações podem ser bastante úteis para o estudo de hábitos e rotinas de habitantes das cidades. Em especial, é possível ressaltar a importância destas informações para os ITS (Intelligent Transportation System), uma vez que considerando os aspectos de mobilidade dos veículos e as suas trajetórias diárias, é possível extrair desse cenário diversos padrões de movimentação e também identificar pontos onde esta mobilidade está fluindo ou precisa ser revista. Com essas informações também é possível detalhar características culturais sobre as rotinas dos usuários, seus interesses e os pontos de maior visitação em uma cidade.

A segunda etapa (ii) análise dos dados urbanos está posicionada após a coleta dos dados. Nessa etapa ocorre o processamento e a combinação de diferentes tipos de dados com o objetivo de extrair um conjunto de informações. Além disto, os dados brutos provenientes da etapa de sensoriamento requerem um processo de preparação dos dados, antes de prosseguir para a etapa de análise dos dados. Um dos grandes desafios nessa etapa é a elaboração e a implementação de um sistema capaz de analisar as correlações de um grande conjunto de dados, extrair informações e disponibilizá-los em tempo quase real. Assim como na primeira etapa, outro desafio que ainda persiste é em relação à fontes de

dados heterogêneas, onde são necessários métodos e ferramentas para realizar a integração de diferente tipos de dados que podem ter sido obtidos das mais diversas fontes.

A terceira etapa (iii) desenvolvimento de serviços e aplicações para cidades inteligentes se refere aos procedimentos adotados para o consumo das informações. Para que esta atividade seja bem-sucedida é necessário o desenvolvimento de modelos descritivos e preditivos a partir das informações adquiridas. Os modelos descritivos podem ser utilizados para identificar relações entre as diversas variáveis coletadas, ou seja, ele serve para descrever o passado e auxiliar na tomada de decisões sobre o futuro. Esse modelo é muito útil para categorizar as informações e também identificar potenciais situações que podem ser ajustadas para prover benefícios aos serviços e/ou seus utilizadores. Uma das desvantagens dos modelos descritivos é que a intuição e a experiência acabam por te um peso excessivo na tomada de decisões.

Devido a computação urbana analisar um grande fluxo de dados, estas características podem levar a resultados tendenciosos que não refletem de forma integral a realidade. Por isto, os modelos preditivos também deverão utilizados. Esses modelos conseguem identificar padrões ocultos e determinas as possíveis consequências resultantes desses padrões. Com este conhecimento em mãos, as decisões tomadas se tornarão mais assertivas. De uma forma geral, os modelos preditivos podem ser desenvolvidos através de métodos supervisionados ou não supervisionados no contexto de Aprendizado de Máquinas. No primeiro, existe uma fase de treinamento do modelo, ou seja, são utilizados dados de entrada e de saída que descrevem a situação em particular que o método está sendo treinado para identificar. Por outro lado, nos modelos não supervisionados, somente os dados de entrada são introduzidos, sendo que o modelo deverá descobrir a relação entre estes dados para apresentar uma saída.

Como a computação urbana é bastante abrangente, neste capítulo vamos nos ater aos aspectos relacionados à computação em nuvem e névoa (Seção 2.2.1), gerenciamento e localidade dos dados (Seção 2.2.2), sensoriamento e aquisição de dados urbanos (Seção 2.2.3) e por último, aspectos relacionados ao *crowdsourcing* e *crowdsensing* (Seção 2.2.4).

2.2.1. Computação em Nuvem e Névoa

A computação em Nuvem proporcionou grandes avanços nas formas de utilização das infraestruturas de Internet. Os seus centros de processamento de dados são, de uma maneira geral, amplas instalações em locais selecionados especificamente para esta finalidade, sendo que na sua maioria, encontram-se distribuídos através do globo terrestre. Cada centro de dados consolida um vasto número de equipamentos em um mesmo local físico, que pode oferecer serviços de forma isolada ou se comunicar com outros centros de dados para melhor atender o usuário final. A centralização destes recursos permite realizar uma economia financeira devido a escala de operação, viabilizando reduzir o custo de funcionado associado à sua instalação, manutenção e gerenciamento.

Um funcionalidade importante da nuvem é a possibilidade de virtualização de equipamentos e serviços. Com isto, é possível prover uma maior elasticidade na alocação de recursos. Por conseguinte, os utilizadores podem requisitar somente a capacidade necessária para a realização das suas operações. Não existe a necessidade de efetuar um

aprovisionamento maior do que o necessário, pois sempre é possível solicitar mais recursos caso haja necessidade.

A convergência das redes de computadores com a nuvem auxiliou na resolução de diversos problemas, como por exemplo, provendo uma maior escalabilidade e disponibilidade, bem como aumentando a compatibilidade entre sistemas. Porém, ao mesmo tempo, introduziu novos desafios provenientes do distanciamento entre a produção e processamento dos dados, levando a uma sobrecarga do núcleo da rede e uma maior latência de comunicação [Curado et al. 2018].

Pelo fato de prover um serviço ubíquo, os seus usuários estão dispersos no mundo todo, por conseguinte, muitos destes se encontram geograficamente distantes dos provedores da nuvem. Esta característica pode não ter grande impacto para o usuário convencional, que utiliza apenas os recursos mais simples disponíveis, como por exemplo, armazenamento de arquivos, redes sociais e navegação na Internet. Por outro lado, isto pode representar um grande desafio para dispositivos restritos, que possuem pouca capacidade de memória, processamento e comunicação, características estas de uma considerável parte dos dispositivos pertencentes a IoT (Internet of Things) que são amplamente usados em cidades urbanas e inteligentes.

Dependendo da aplicação, mesmo em dispositivos de maior capacidade, esta distância pode se tornar um impedimento para atingir plenamente os requisitos dos serviços disponibilizados, como por exemplo, em previsão de congestionamento e gerenciamento de energia. Desta forma, é necessário trazer ao menos uma parcela do poder computacional mais próximo dos usuários e/ou serviços. Por isto, a computação em névoa vem ganhando força [Byers 2017]. A proposta básica deste paradigma é trazer a elasticidade de recursos da nuvem para perto dos usuários/objetos/serviços.

A utilização conjunta dos paradigmas de nuvem e névoa pode ser organizada em uma arquitetura de camadas, conforme definido pelo consórcio Openfog [Group et al. 2017] e ilustrado na Figura 2.2. Um dos modelos aceitos hoje em dia é formado por três camadas, que são: a camada ciber-física, camada da névoa e a camada da nuvem. A camada inferior é composta pelos dispositivos físicos, sejam eles móveis ou estáticos. Com o aumento do poder computacional destes equipamentos e o aprimoramento das tecnologias de comunicação, estes são capazes de executar processamentos relativamente complexos com uma boa performance [Taleb et al. 2017a].

Estes aparelhos, como por exemplo *smartphones*, *smartwatches*, consoles de jogos, óculos inteligentes, entre outros, em conjunto com os demais servidores e equipamentos, da borda da rede, localizados próximos aos usuários, podem ser utilizados para aliviar o ambiente da nuvem e ainda prover menor latência na comunicação.

A camada intermediária (névoa), é composta por múltiplos níveis, com o objetivo de fornecer uma maior flexibilidade na composição da sua infraestrutura. Sendo assim, com o estabelecimento desta hierarquia de níveis é possível oferecer um equilíbrio em algumas das características mais desejáveis, como a capacidade computacional, tempo de acesso e responsabilidade organizacional. De uma forma geral, os níveis mais próximos da borda da rede tendem a possuir uma menor responsabilidade organizacional na infraestrutura e, portanto, uma menor cobertura geográfica. Por outro lado, níveis mais próximos da

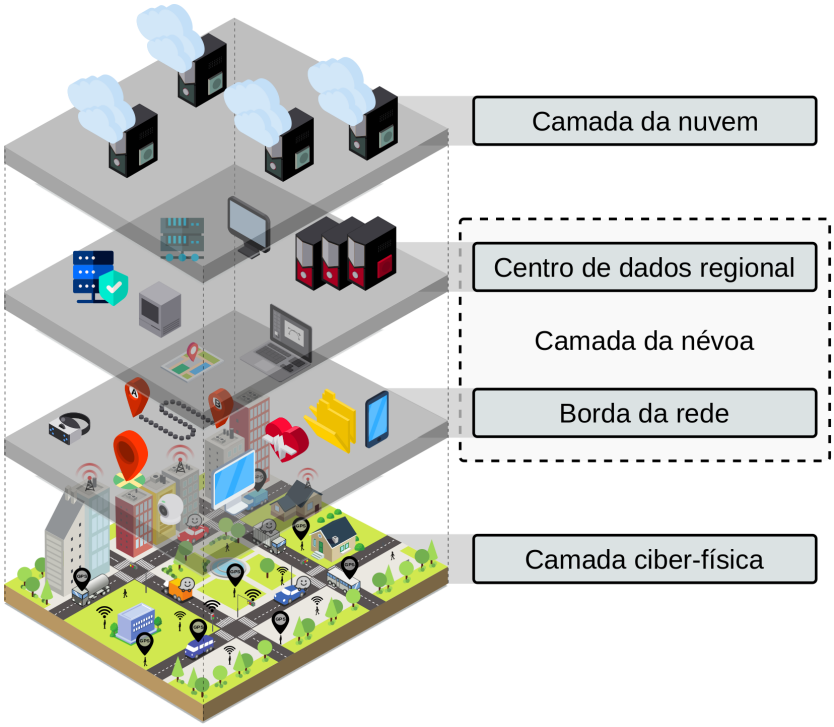


Figura 2.2. Arquitetura de computação urbana com múltiplas camadas de névoa e nuvem

nuvem possuem uma maior cobertura geográfica e capacidade computacional, mas tendem a apresentar tempos de acesso mais elevados. O camada superior é composta pelos servidores de nuvem, os quais podem estar localizados em nuvens públicas, privadas, ou uma combinação de ambas.

É válido mencionar que um dos maiores desafios em uma rede centrada na computação em névoa é como alcançar o melhor desempenho possível considerando o poder de processamento e comunicação de cada equipamento, em conjunto com o processamento e comunicação oferecidos pela névoa. Isto deve-se ao fato de que são esperados dispositivos com características heterogêneas nestas redes. A computação em névoa precisa levar em consideração a heterogeneidade tanto das tecnologias de rede como dos dispositivos, permitindo que estes coexistam e operem em conjunto para prover níveis satisfatórios de qualidade. Por isto, um amplo conjunto de parâmetros devem ser contemplados, como por exemplo, dispositivos móveis com capacidade de bateria limitadas, diferentes formas e capacidades de comunicação e também distintas capacidades de processamento [Bittencourt et al. 2018].

Neste sentido, um aspecto importante é a Qualidade de Experiência (QoE) dos usuários finais. Este conjunto de métricas busca definir de uma forma quantitativa a aceitação de um serviço/aplicação, considerando desde a origem das informações até como elas são apresentadas na sua forma final. Por isto, para uma solução ser considerada adequada para a computação em névoa, esta também deve levar em consideração a expectativa de QoE dos usuários finais e o seu impacto no sistema como um todo.

Além de desafogar o ambiente de nuvem e oferecer comunicação de baixa latência, a computação em névoa também pode prover serviços cientes de sua geolocalização. Tendo em vista que o processamento das informações será realizado na borda da rede, ou seja em

uma localização geográfica muito próxima ao usuário, diversos detalhes sobre a rede de acesso podem ser utilizadas, como a qualidade de sinal da borda até o usuário e também a carga atual da célula em que o usuário se encontra. Todas estas informações podem se apresentar como um grande diferencial para a maioria das aplicações de computação urbana.

2.2.2. Gerenciamento e Localidade dos dados

Conforme mencionado anteriormente, a produção e consumo de dados está atingindo números recordes e continua com ritmos de crescimento sem precedentes. Como resultado desta tendência, o gerenciamento e localidade dos dados vêm ganhando muita atenção ultimamente. De uma forma simplificada, estes conceitos se referem à capacidade de organizar e manter processos relacionados a aquisição, processamento, distribuição e armazenamento de dados, bem como a proteção e validação de informações. Esses conceitos foram pesquisados no passado em diferentes contextos, como por exemplo, em *clusters* computacionais e também nas áreas de computação paralela e distribuída. No entanto, até recentemente estes princípios ainda não haviam sido aplicados em centro de dados geograficamente distribuídos, como os presentes na computação em nuvem e névoa [Xu and Li 2013, Hung et al. 2015, Heintz et al. 2016].

O gerenciamento de dados faz alusão ao planejamento e implantação de uma série de políticas e procedimentos para realizar a gestão completa, precisa e integral do ciclo de vida dos dados. De uma forma geral, esta atividade pode ser dividida em duas estratégias distintas, as quais definem (i) o posicionamento e (ii) a forma de acesso [Sakr et al. 2011]. A estratégia de posicionamento busca especificar onde estas informações serão armazenadas e como serão disseminadas. Esse procedimento, por exemplo, é responsável por determinar quantas cópias dos dados devem ser realizadas e também quais são os melhores locais/dispositivos para armazenar essas informações. A segunda estratégia define como deverão ser tratadas pelo sistema as operações de leitura e escrita. Para esta finalidade, é necessário considerar conjuntamente os procedimentos que deverão manter a consistência dos dados entre cópias distribuídas. Outra responsabilidade é também a de definir as formas de acesso as informações replicadas.

Assim como o gerenciamento de dados, o conceito de localidade é igualmente importante. Este pode ser definido como a habilidade de mover ou alocar a capacidade computacional necessária para o processamento de informações nos locais próximos de onde estão sendo criados e/ou adquiridos estes dados [Yang et al. 2017]. Desta forma, é possível evitar a transferência indiscriminada de uma grande quantidade de dados brutos diretamente até recursos computacionais centralizados, como por exemplo, servidores da nuvem. Esta estratégia baseia-se no fato de que movendo e executando os aplicativos computacionais para próximo dos dados sobre os quais eles operam é possível realizar uma computação mais eficiente e ainda prover a economia de recursos de rede. Com esta estrutura, consequentemente, elimina-se a necessidade de transmitir grandes quantidades de dados não processados até o aplicativo/serviço. É importante ressaltar que está estratégia é válida para serviços que necessitam de uma grande quantidade de dados para operar ou, o quais precisam fornecer uma resposta ultrarrápida as solicitações dos usuários. Por este motivo, os fundamentos da localidade de dados se encaixa com uma das principais premissas da computação em névoa, que é ter um sistema descentralizado com recursos

próximos aos usuários finais [Wen et al. 2017].

Nesta perspectiva, pesquisas confirmam que o armazenamento de dados nas bordas da rede, utilizando a computação em névoa, pode melhorar o tempo de resposta e reduzir o tráfego de rede [Confais et al. 2017]. Além das questões de desempenho, a localidade dos dados pode aprimorar mecanismos relacionados à segurança e privacidade das informações [Bellavista and Zanni 2017]. Operando localmente, é possível ter um conhecimento preciso da estrutura da rede e implementar mais facilmente recursos de autenticação e autorização. Outra vantagem de ter os dados próximos de onde estes serão processados é o aperfeiçoamento das questões de privacidade [Vaquero and Rodero-Merino 2014], pois as informações não precisam ser transferidas através da rede, ou seja, existe um maior controle de acesso sobre as mesmas.

Outra vantagem da localidade dos dados é a possibilidade de fazer o descarregamento de operações, tais como processamento ou armazenamento. Mesmo com o avanço nas tecnologias móveis, em algumas situações ainda é vantajoso transferir o processamento do próprio dispositivo para a borda da rede visando uma economia de bateria e a redução no tempo de processamento [S. Gama et al. 2018]. A computação em névoa auxilia nesta atividade provendo camadas de recursos entre os usuários e a nuvem [Yi et al. 2015, Shi et al. 2016]. É importante ressaltar que com a dispersão dos recursos através de toda a hierarquia de rede, são necessários novos mecanismos de gerenciamento de recursos, bem como métodos de migração para garantir que as aplicações e dados descarregados estão sempre o mais próximo possível dos usuários [Taleb et al. 2017b].

Na computação em nuvem, a migração de máquinas virtuais, serviços e dados pode ser utilizada para realizar o balanceamento de carga de centro de dados e também para consolidar estas operações em uma quantidade menor de servidores. De mesma forma, na computação em névoa, a migração destes serviços/aplicações pode ser utilizada para replicar e/ou mover tanto dados como poder de computação para próximo dos usuários que necessitam ou poderão necessitar em um futuro iminente. Através da computação urbana, por exemplo, pode ser identificado os principais locais onde os usuários costumam realizar acessos e consumir informações. Tendo em vista que estes padrões de comportamento da mobilidade humana podem ser previsto de uma forma consistente [Song et al. 2010], é possível carregar antecipadamente os dados e serviços nesses locais, reduzindo consideravelmente o tempo de acesso aos mesmos [Bittencourt et al. 2015].

2.2.3. Sensoriamento e aquisição de dados urbanos

A extração de conhecimento a partir dos dados consiste, basicamente, na modelagem e análise dos dados definindo uma semântica para a tomada de decisão para um determinado serviço [Barnaghi et al. 2012], por exemplo, a definição de rotas mais rápidas e seguras. A aquisição e processamento de dados tem sido um dos grandes desafios em Computação Urbana, pois os dados são heterogêneos, provem de diversas fontes, possuem diferentes formatos (tabelas, mapas, grafos, etc) e são de diferentes tipos (dados sobre o tráfego, qualidade do ar, dados sociais e dados geográficos, etc). De acordo com a Figura 2.1 as etapas necessárias para o tratamento dos dados até que eles estejam disponíveis para a aplicação são a coleta dos dados, processamento, armazenamento e modelagem. Estas etapas são detalhadas a seguir.

A *coleta de dados* é a primeira etapa a ser realizada. De acordo com [Silva et al. 2018], os dados usados nessa área são provenientes de quatro fontes:

- Dados de sensores: atualmente, as redes de sensores são muito utilizados para coleta de dados em ambientes urbanos. Elas fornecem dados que são obtidos através da instalação de sensores dedicados a aplicações específicas, por exemplo, sensores para monitoramento da qualidade do ar em diversos pontos da cidade ou sensores para o monitoramento de níveis de ruídos. Atualmente, é muito comum o usos de veículos conectados e dispositivos móveis para a obtenção de dados. Nesse caso, deve-se considerar o custo existente para a construção da redes de sensoriamento e que os veículos devem estar previamente equipados.

- Dados da infraestrutura disponível nas cidades: atualmente as cidades estão construídos infraestruturas para diversos propósitos, por exemplo a rede de telefonia celular, redes sem fio ou ainda sistemas de transporte público. Muitas vezes, essa infraestrutura também pode ser utilizada para coletar dados.

- Dados estatísticos: são dados referentes a estudos estatísticos sobre uma população, tais como dados demográficos, econômicos e sociais relativos a um momento determinado ou em certos períodos. O trabalho de [de Souza et al. 2018] implementa uma ferramenta para sugestão de rotas usando, além de dados sobre a condição de tráfego, dados relacionados a crimes na cidade de São Paulo. Os dados foram obtidos do banco de dados oficial do departamento de policia. Vale ressaltar, que muitas vezes não existe a disponibilidade desses dados e quando disponíveis eles possuem diversos formatos ou incompletos. Assim, deve existir um mecanismo para obtenção e tratamento desses dados antes de sua utilização.

- Dados de redes sociais: no ambiente de computação urbana os dados das mídias sociais, tais como Twitter, *Instagram*, *Waze* e *Foursquare* são usados para modelar a mobilidade das pessoas nas áreas urbanas e são usados em diversas aplicações, tais como na identificação de problemas no trânsito causados por acidentes, protestos, desastres, etc [Silva et al. 2013, Ribeiro et al. 2014, Santos et al. 2018, Rodrigues et al. 2018]. Esses dados possuem diversos formatos, tais como textos, fotos e vídeos. Aqui está incluído, o sensoriamento participativo, que é o processo pelo qual indivíduos e comunidades usam celulares e serviços de nuvem para compartilhar voluntariamente informações [Estrin et al. 2010]. Vale destacar que a obtenção desse tipo de dado pode ser mais fácil do que os três anteriores, dada a grande quantidade desses dados disponíveis na WEB. Esses dados podem ser obtidos por meio de APIs disponibilizadas pelas ferramentas; através de *Web crawler*, que são programas que analisam páginas Web em busca de dados relevantes; ou desenvolvendo aplicações em plataformas já existentes, por exemplo, algumas ferramentas, como Facebook e Instagram, permitem a criação de aplicativos dentro de suas plataformas.

Dessa forma, existem diversas forma de coleta de dados no ambiente de computação urbana, nesse minicurso serão usados dados de redes sociais para exemplificar o desenvolvimento de uma aplicação no ambiente de computação urbana.

Uma vez que esses dados são coletados, deve-se utilizar técnicas de *processamento e armazenamento*, já que essa etapa envolve a manipulação de um grande volume de dados. Em alguns casos, os dados coletados são suficientes para revelar um padrão interessante, não sendo necessário um processamento complexo desses dados. No entanto, na maioria das vezes esses dados são provenientes de diversas fontes e possuem formatos diferentes, dessa forma são necessários técnicas específicas para formatação desses dados. Por exemplo, no problema de geração de rotas em ambientes urbanos podemos considerar as informações de trânsito disponibilizadas pela infraestrutura da cidade e informações a respeito de áreas de perigo. Dependendo da aplicação, será necessário a comparação dos dados atuais com tendências passadas, de modo que o armazenamento e o gerenciamento robusto e de longo prazo desses dados são um requisito central. De uma forma geral, nessa etapa são desenvolvidos *scripts* para a formatação e organização dos dados. A organização dos dados nos leva a entender melhor esses dados e definir quais os modelos e análise são apropriados para serem aplicados [Silva et al. 2018]. Na Seção 2.5 será mostrado um exemplo prático do processamento de um tipo de dados específico.

O armazenamento envolve sistemas de arquivo e ferramentas para processamento distribuído de grande volumes de dados e deve ser: escalável, distribuído, seguro, tolerante a falhas e consistente. No caso da computação em névoa, os dados são movidos para o melhor lugar para processamento, isto é, mais perto do limite da rede. Portanto é uma arquitetura de computação descentralizada onde dados, cálculos, comunicações, armazenamentos, medições, aplicações e gerenciamentos são distribuídos entre a fonte de dados e a nuvem. Dessa forma, tem-se uma redução de dados que necessitam ser transportados para análise, processamento ou armazenamento e maior velocidade no acesso aos dados pelo usuário final.

Algumas técnicas são utilizadas para *modelagem* dos dados. Os dados aqui possuem propriedades tanto espaciais, como temporais. Por exemplo, a localização de estabelecimentos é um dado espacial; dados meteorológicos, vídeos de vigilância e consumo de energia são dados temporais. Já os dados como como fluxos de tráfego e mobilidade humana possuem propriedades espaço-temporais simultaneamente [Zheng et al. 2014]. Nessa etapa os dados devem ser organizados em alguma estrutura que incorpora simultaneamente o dado e as informações espaço-temporais para suportar uma análise de dados eficiente. Além disso, um sistema de computação urbana geralmente precisa aproveitar uma variedade de dados heterogêneos e responder rapidamente às consultas dos usuários, por exemplo, as condições de tráfego em uma determinada área. Levando em conta as propriedades espaço-temporais desses dados, temos que encontrar modelos que consigam capturar a dinâmica dos dados. Em computação urbana, é comum a utilização de grafos para representar a propriedade espacial. Por exemplo, em uma rede veicular mapeada como um grafo, cada veículo representa um nó do grafo e as ligações são estabelecidas com o nó se dois veículos conseguem se comunicar. O conteúdo desses dados pode variar muito ao longo do tempo, assim um modelo baseado em grafos estáticos não é suficiente para capturar essa dinamicidade, dessa forma a teoria de Redes Complexas tem sido utilizada para representar diversos problemas nesse contexto. Essa teoria provê modelos (redes de mundo pequeno, redes livres de escala) e métricas (p.e., centralidade de nós, coeficiente de agrupamento) que auxiliam no entendimento da dinâmica do problema [Naboulsi and Fiore 2013, Moura et al. 2018].

A seguir pode ser realizada a análise de dados para se identificar anomalias, tais como os locais onde a mobilidade das pessoas difere significativamente de seus padrões de origem. Dessa forma, vamos caracterizar os dados coletados, a fim de entender suas limitações e sua utilidade. Geralmente essa etapa é composta pelas etapas de extração e análise da informação obtida a partir dos dados brutos e validação de resultados. A extração de conhecimento pode explorar diferentes abordagens, dependendo do problema que estamos tentando resolver. Um procedimento típico realizado é a investigação preliminar dos dados para entender suas propriedades, auxiliando na escolha de técnicas de análise.

As técnicas de visualização, como histogramas e diagramas de dispersão, medidas estatísticas, como média e desvio padrão de um conjunto de valores, são métodos comuns usados para explorar propriedades de dados nesta investigação preliminar [Silva et al. 2018].

Algumas tecnologias e ferramentas devem ser utilizadas nessa etapa para a análise dos dados. Técnicas como, algoritmos de agrupamento, regressões, modelos probabilísticos e análise de sentimento são muito utilizadas nesse contexto. Em relação as ferramentas para análise, as linguagens Python, Matlab/Octave e R são robustas, eficientes e de fácil compreensão para o desenvolvimento dos algoritmos.

2.2.4. Crowdsourcing e Crowdsensing

Conforme mencionado anteriormente, uma das principais funções da computação urbana é referente ao processo de aquisição, integração e análise de grande volume de dados. Na sua essência, estes dados costuma ser heterogêneos, ou seja, não seguem nenhum padrão definido, e gerados por d fontes em diferentes espaços urbanos. Além das tecnologias, modelos analíticos e métodos aplicados na Computação Urbana, dois importantes paradigmas podem ser também aplicados: *crowdsourcing* e *crowdsensing*. A seguir uma breve explicação de cada uma deles.

Crowdsensing móvel é um paradigma que utiliza o conceito de computação ubíqua no sensoriamento do ambiente, além de compartilhar os dados sensoriados. [Wang et al. 2018, Akabane et al. 2018a, Akabane et al. 2018b]. Na primeira camada da arquitetura da Figura 2.3 apresenta um exemplo de *crowdsensing*, onde diferentes dispositivos são utilizadas para extrair coletar dados brutos. Além disso, por meio da agregação de tais dados, pode-se criar uma consciência local que pode auxiliar aplicações de larga escala, por exemplo, monitoramento da poluição do ar e alerta de congestionamento de tráfego veicular. Sabe-se que nas *Vehicle Social Networks* - VSNs, os veículos são equipados com tecnologias de computação e de comunicação sem-fio juntamente com sensores inteligentes, assim possibilitando o surgimento do paradigma de *vehicle crowdsensing* - VCS [Akabane et al. 2018a]. Tal paradigma possibilita o monitoramento de fenômenos dinâmicos e em grande escala. A motivação do uso do *crowdsensing* móvel está no fato de que os participantes possam mapear fenômenos de interesse comum. Por exemplo, os participantes de uma VSNs podem conjuntamente monitorar as condições do tráfego veicular, e também, aprimorar a mobilidade urbana por meio do compartilhamento dos dados sensoriados. Ao fazer isso, aplicações de VSNs podem agregar os dados locais sensoriados e gerar conhecimento sobre as condições de tráfego em tempo-real, e tal conhecimento auxiliará o gerenciamento da mobilidade urbana. Outro exemplo [Issarny

et al. 2018] é o monitoramento da exposição à poluição em ambiente urbano.

O *crowdsourcing* é um paradigma de cooperação, no qual um grande grupo de usuários (sendo que cada usuário resolve pequenas sub-tarefas de um trabalho maior) trata de problemas complexos que podem ser resolvidos, por meio da participação de muitos usuários, de maneira eficiente, por exemplo, gerenciamento inteligente de tráfego veicular [Akabane et al. 2018a]). Na camada do meio arquitetura da Figura 2.3 apresenta um exemplo de *crowdsourcing*, onde diferentes tecnologias são empregadas para extrair conhecimento dos dados sensoriados. Outra ideia muito aplicada é a utilização de sensores móveis no contexto de *crowdsourcing*, denominado de *Mobile Crowdsourcing* (MCS). O MCS difere das redes de sensores tradicionais, pois este envolvem pessoas em movimento que estão processando dados ao seu redor de diferentes locais. Neste caso, as pessoas não estão apenas transportando sensores integrados aos seus dispositivos móveis (*smartphones*), mas também são capazes de fornecer informações manualmente, como fotos e vídeos. As vantagens do MCS em relação a outros tipos de redes de sensores incluem: (i) alta capacidade de computação dos dispositivos móveis para pré-processamento de dados; (ii) conectividade com nuvem ou névoa; e (iii) usuários podem fornecer informações do ambiente que não são coletadas pelos sensores. Com isso surgem sistemas que identificam anomalias urbanas baseado em *crowdsourcing*. Tais sistemas permitem a comunicação pervasiva e em tempo-real das anomalias que ocorrem na cidade (por exemplo, ruído, uso ilegal de instalações públicas, mau funcionamento da infra-estrutura urbana).

Vale salientar que vários sistemas e aplicativos de *crowdsourcing* e *crowdsensing* desenvolvidos no paradigma da Computação Urbana são dedicados a permitir que os cidadãos colaborem na melhoria da qualidade de vida no ambiente urbano.

2.3. Aplicações

Esta seção apresenta algumas das principais aplicações e serviços no contexto de computação urbana em diferentes domínios. O objetivo é mostrar que a integração e análise de dados obtidos de diferentes fontes em espaços urbanos têm um papel importante em prol do desenvolvimento das cidades inteligentes. Com isso, a Seção 2.3.1 apresenta os sistemas de transportes inteligentes. A Seção 2.3.2 discute aplicações e serviços sociais baseadas em localização. Por fim, a Seção 2.3.3 apresenta as aplicações no contexto de segurança pública.

2.3.1. Sistemas de Transportes Inteligentes

O setor de transporte é um dos principais propulsores para desenvolvimento econômico de uma nação. Tanto cidadãos quanto empresas dependem desse setor no dia a dia. Por exemplo, as empresas dependem dos meios de transporte para obter os insumos, além de distribuir seus produtos até os consumidores. Já os cidadãos necessitam de tais meios para o deslocamento diário da casa para o trabalho e vice-versa.

Com os avanços das tecnologias de comunicação sem fio e da computação móvel permitiram o surgimento dos sistemas de transporte inteligentes (ITS). Tais sistemas têm como objetivo principal melhorar a mobilidade urbana, além de aumentar a segurança no trânsito. Aplicações desenvolvidas para ITS, geralmente, visam auxiliar os condutores e os passageiros durante seus deslocamentos, com intuito de reduzir acidentes (*aplicação*

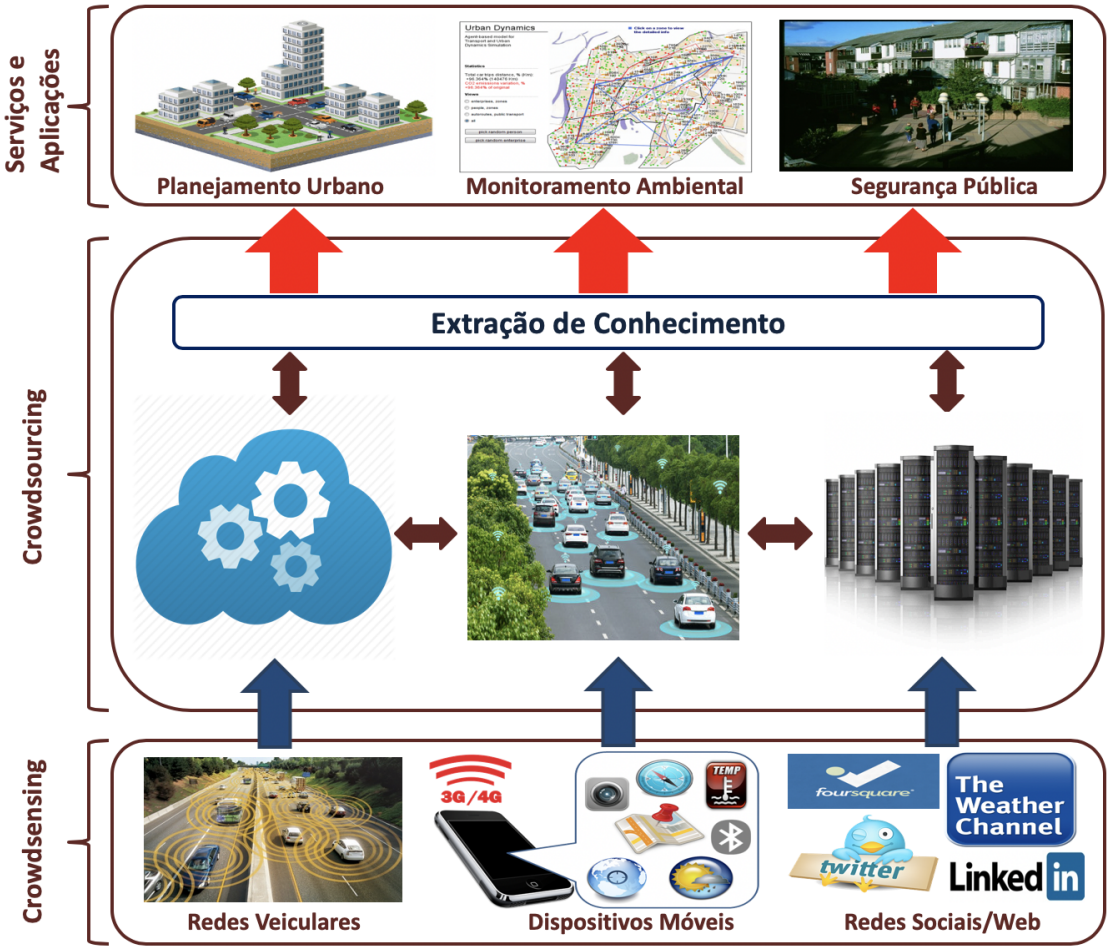


Figura 2.3. Uma arquitetura ilustrativa de *crowdsourcing* e *Crowdsensing*.

de segurança) e melhorar o gerenciamento de tráfego veicular (*aplicação de eficiência de tráfego*) [Meneguette et al. 2016b, Meneguette et al. 2016a]. Além dessas, há outro tipo, *aplicação de entretenimento e conforto*, como o próprio nome sugere, esta tem o objetivo de tornar a viagem mais prazerosa e tranquila. Para mais detalhes de cada aplicação mencionada anteriormente, consulte [Cunha et al. 2017].

Outra aplicação de ITS é o *transporte multimodal* (TM), foco do estudo de caso deste minicurso (apresentado na Seção 1.5). A Figura 2.4 apresenta um exemplo ilustrativo de TM, no qual utiliza diversas modalidades de transporte (por exemplo: táxi, ônibus, trem ou metrô) para transportar cargas ou passageiros entre os pontos de origem e de destino [Zografos and Androutsopoulos 2008, SteadieSeifi et al. 2014]. Por consequência, o TM oferece um serviço que permite o transporte mais eficiente, confiável, flexível e sustentável. Sabe-se que, muitos cidadãos dos grandes centros urbanos necessitam utilizar diferentes tipos de transporte público multimodal nos seus deslocamentos diários. Logo, encontrar o caminho com menor custo, entre dois pontos, levando em consideração mais de um modo de transporte é o principal desafio do Problema de Transporte Multimodal.

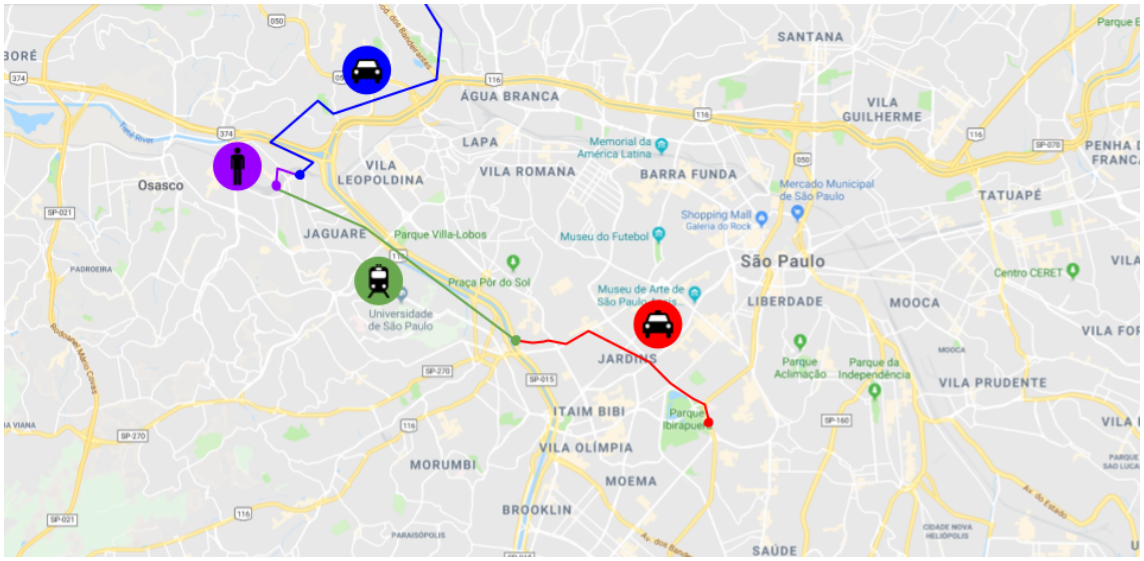


Figura 2.4. Um exemplo ilustrativo de transporte multimodal.

Alguns trabalhos da literatura lidam com esse tipo de problema. Por exemplo, no trabalho de [Duque et al. 2015], foi aplicado um algoritmo exato para a seleção dos caminhos. Para isto, duas variáveis conflitantes, custo e tempo, são aplicadas na solução. O algoritmo proposto foi inspirado no "Algoritmo de Pulso" [Birari and Iyer 2005], ou seja, envia um pulso na rede e mensura o caminho realizado por ele, em relação aos pesos dos arcos até ao destino. Utilizando as mesmas variáveis conflitantes, no trabalho de [Sedeno-Noda and Raith 2015] foi aplicado um algoritmo de caminho mínimo e a solução é classificada por meio do conceito da não dominância de uma das variáveis.

No trabalho de [Idri et al. 2017], os autores propõem um algoritmo "orientado ao alvo". Tal algoritmo aplica a busca pela proximidade dos nós em relação ao destino, ou seja, nessa etapa constrói-se o caminho virtual. O cálculo da proximidade é feito a partir da distância Euclidiana entre os pontos de origem e de destino. Além disso, durante o cálculo é aplicado o conceito de desigualdade triangular para priorizar a escolha dos nós

localizados mais próximo do destino. Outra abordagem concebida pelos autores [Dib et al. 2015] é combinar duas meta-heurísticas (Algoritmos Genético e de Busca Local), com intuito de simular o Algoritmo Memético [Krasnogor and Smith 2005]. Tal algoritmo, faz uma analogia à memória humana, nas quais são memorizadas/armazenadas as informações mais importantes do processo, para que no final seja possível recuperar tais informações.

2.3.2. Aplicações sociais baseadas em localização

Atualmente existem diferentes tipos de redes sociais na Internet com diversos propósitos. Nesta seção, nosso foco é as redes sociais baseadas em localização (LBSN), um tipo especial de rede social em que informações geolocalizadas representam um papel fundamental no sistema. Estudar LBSNs é interessante, pois elas geram um tipo específico de dados digitais que oferecem resoluções geográficas e temporais sem precedentes. Com isso, os dados provenientes desses sistemas podem ser úteis para entender diversos fenômenos urbanos em larga escala e com um custo, potencialmente, mais baixo [Silva et al. 2018]. O Foursquare, Waze e Instagram são exemplos clássicos de LBSNs, no entanto existem vários outros exemplos um pouco menos conhecidos, como o Untappd[1] que é uma rede especializada cervejas.

De acordo com [Zheng 2011] uma rede social baseada em localização consiste em uma nova estrutura social composta por indivíduos ligados pela interdependência derivada de suas localizações físicas, bem como o seu conteúdo (como fotos, vídeos e textos) geolocalizados. A interdependência entre dois indivíduos existe, por exemplo, tem históricos semelhantes de localização e possuem interesses em comum (i.e., comportamentos e atividades).

As LBSNs contam com dados geográficos em (quase) tempo real coletados de forma voluntária com o auxílio de dispositivos móveis dos indivíduos. Esse é um dos fatores que favorecem a alta escalabilidade dessas redes [Silva 2016]. Esses dados podem ser úteis para prover, por exemplo, entretenimento e informações de tipos variados (e.g., para tomar melhores decisões no trânsito).

Dados provenientes de LBSNs permitem entender usuários e locais no mundo físico, bem como explorar a relação entre eles. Para a estimativa de semelhança entre usuários, etapa que é útil em diversas aplicações e serviços, é importante descrever o comportamento e interesses dos usuários. Existem várias maneiras para estimar a semelhança entre usuários, e, em geral, uma boa estimativa de semelhança depende da disponibilidade de boas informações descritivas, tais como visitas de locais (no geral, quanto mais melhor), a popularidade de locais, bem como a granularidade geoespacial das informações [Zheng et al. 2014]. O comportamento e interesses de um indivíduo podem descritos, por exemplo, pelo seu histórico de *check-ins*[2] na cidade. Assim, pessoas que compartilham informações semelhantes de localização, i.e., visitam locais semelhantes, tendem a ter interesses e comportamentos comuns. A partir dessas informações pode-se detectar comunidades que representam um grupo de pessoas com interesses comuns, para, por exemplo, a realização propagandas direcionadas, bem como criar sistemas de recomendação para diversos tipos de conteúdo.

[1] https://untappd.com.

[2] ato de disponibilizar a sua localização atual para amigos na rede.

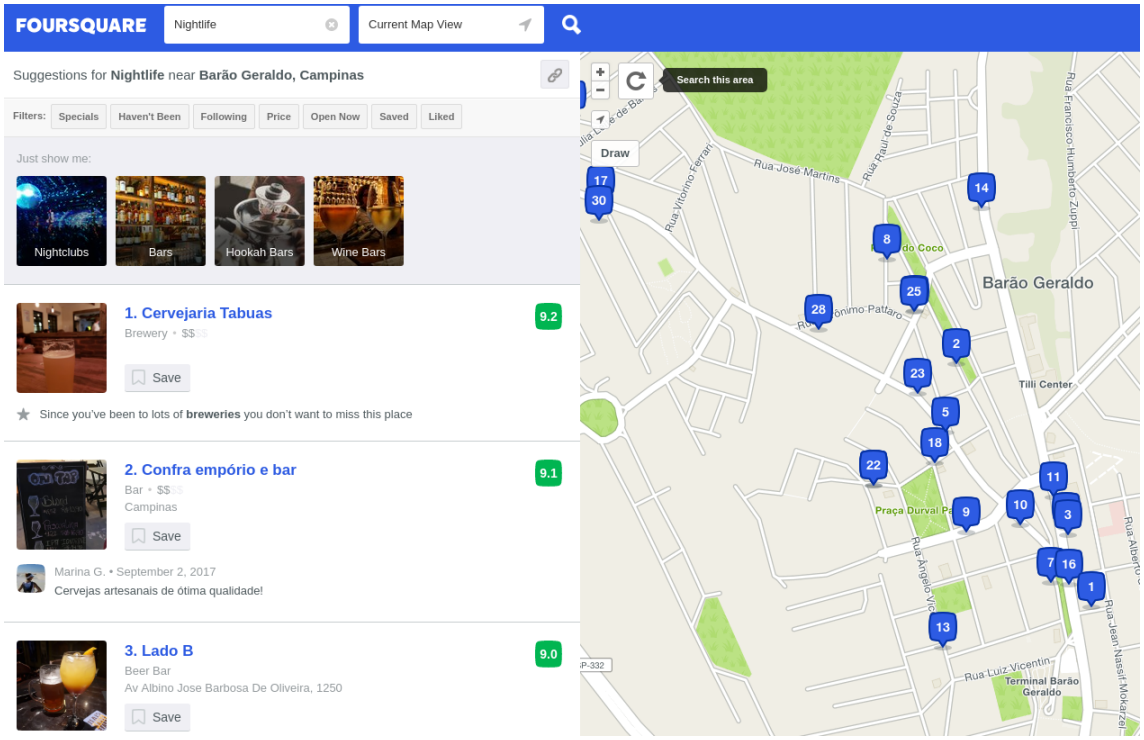


Figura 2.5. Um exemplo ilustrativo de recomendação de locais fornecida pelo Foursquare.

Um recomendador de locais é um exemplo específico de aplicação de recomendação que consiste em encontrar e recomendar os locais mais interessantes em uma cidade. Vários fatores precisam ser levados em consideração para oferecer uma boa recomendação de locais. O nível de interesse de um local não depende exclusivamente do número de pessoas que visitaram o mesmo. Por exemplo, o local mais frequentemente visitado em uma cidade pode ser uma estação rodoviária ou aeroporto, mas esse tipo de local tende a não ser uma localização interessante de ser recomendada, por exemplo, a um turista que está visitando uma cidade. Com isso, além da popularidade, é importante levar em conta a semântica dos locais, que pode ser obtida com a descrição disponível nas LBSNs, bem como o perfil dos indivíduos que frequentam o local, que poderia ser obtido, por exemplo, através das semelhanças entre usuários. A Figura 2.5 mostra um exemplo ilustrativo de recomendação de locais fornecida pelo Foursquare. Esse recomendador utiliza, além de outras informações, a opinião dos usuários que já visitaram previamente os estabelecimentos, bem como a popularidade do local e tendências que ocorrem na área estudada [Kim 2015].

Explorando também os dados do Foursquare, Mueller e outros [Mueller et al. 2017] propuseram uma metodologia para caracterizar as preferências de gênero por locais em diferentes regiões e em diferentes granularidades espaciais com base nos dados de LBSNs. Tradicionalmente, os dados utilizados para apoiar esses estudos são obtidos manualmente, muitas vezes por meio de pesquisas com voluntários. No entanto, devido a seus altos custos devido a etapas manuais, esses métodos tradicionais não escalam rapidamente para estudos de grande porte. Os resultados desse estudo sugerem que a metodologia proposta pode ser uma ferramenta promissora para apoiar estudos sobre preferências de gênero por locais ao redor do mundo, sendo mais rápida e barata que os métodos tradicionais.

Um passo fundamental da metodologia é explorar o uso do check-ins do Foursquare para estimar as preferências culturais de gênero por locais no mundo físico. Isso ilustra que os mesmos dados de LBSNs (por exemplo, Foursquare) podem ser usados para diferentes propósitos.

A Figura 2.6 exemplifica o resultado da metodologia proposta, mostrando que ela pode ajudar a identificar as preferências de gênero para locais com granularidades mais finas, por exemplo, considerando todos os locais individualmente para São Paulo. A metodologia proposta identificou 21 locais onde a diferença de popularidade entre gêneros é estatisticamente significativa. Um exemplo é uma universidade privada (que pediu explicitamente para ser anonimizada), que é mais popular entre usuários do sexo feminino. Isso pode ser explicado por uma presença muitas vezes maior de mulheres nos cursos específicos localizados no campus (ou seja, saúde, artes, pedagogia e produção de mídia) no Brasil. Um porta-voz da universidade anônima confirmou, por e-mail, que a maioria pessoas matriculadas no campus são mulheres. Outro exemplo é o Museu de Arte da Fundação Bienal Ibirapuera, que também é significativamente mais popular entre mulheres. Este resultado foi confirmado por dados oficiais [Mueller et al. 2017].

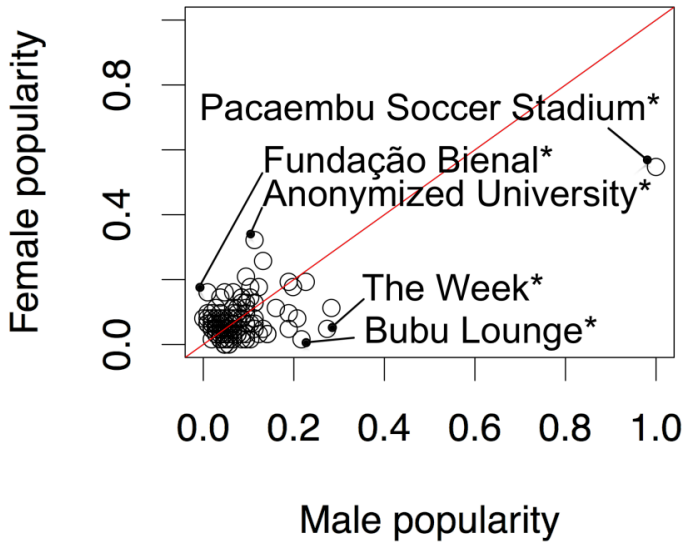


Figura 2.6. Popularidade (normalizada) de locais individuais dentro de cada grupo de gênero em São Paulo, considerando todos os valores de todas as sub-categorias. [Imagem de [Mueller et al. 2017]].

Ao explorar as preferências de gênero por diferenças de locais, os proprietários de empresas poderiam obter informações valiosas sobre seus clientes. Além disso, as recomendações de locais podem se tornar mais conscientes culturalmente, já que homens e mulheres podem ter preferências diferentes em regiões com culturas distintas. Podemos citar ainda que estudos sociológicos sobre preferências de gênero por locais poderiam ser feitos em menos tempo, com tamanhos amostrais maiores e em regiões com granularidades espaciais arbitrárias [Mueller et al. 2017].

Para mais detalhes da utilidade de redes sociais baseadas em localização para área de computação urbana consulte [Silva et al. 2018]. Nesse trabalho os autores discutem conceitos fundamentais da computação urbana utilizando dados do LBSN e apresentam um levantamento de estudos recentes de computação urbana que utilizam dados do LBSN.

2.3.3. Aplicações e serviços em segurança pública

Segurança pública é exemplo de aplicação (ou serviço) desenvolvido no contexto da computação urbana. Sabe-se que alguns eventos observados em grande centros urbanos, por exemplo, acidentes, congestionamentos, inundações e ataques terroristas, representam ameaças que deturpam a ordem das cidades, bem como impor riscos à segurança pública. Além disso, em tais centros, há uma ampla disponibilidade de diferentes tipos de dados que podem ser utilizados pela aplicação. Tanto para lidar corretamente com as ameaças mencionadas anteriormente, quanto para detectar uma ameaça ou até mesmo prevê-las [Riveiro et al. 2017, Wang et al. 2017, Akabane et al. 2018a].

Além da segurança pública, outra aplicação de grande importância dentro da computação urbana é a detecção de anomalias. Uma das anomalias comumente observada em áreas urbanas é de tráfego, particularmente congestionamento. Tal anomalia pode ser causada por diversos fatores, por exemplo, acidentes, protestos, construção e desastres naturais. Assim a detecção ou prevenção de tais anomalias podem ajudar na gestão de tráfego. Algumas soluções podem ser encontradas em trabalhos relacionados que lidam com a detecção de tal anomalia [De Souza et al. 2017, Riveiro et al. 2017, Wang et al. 2017, Akabane et al. 2018a]. Dentre as soluções mencionadas anteriormente, apenas os trabalhos de [De Souza et al. 2017] e [Akabane et al. 2018a] propõem eliminar (ou amenizar) a anomalia de forma reativa. Em outras palavras, uma vez identificada a anomalia de tráfego veicular, cada solução aplica uma estratégia de re-roteamento dos veículos que estão deslocando em direção ao congestionamento. A principal diferença entre as duas soluções é que no trabalho de [Akabane et al. 2018a] a etapa do cálculo da rota alternativa é feito de forma colaborativa. Isto é, tal cálculo é realizado com base nas rotas escolhidas pela vizinhança, além do mais os veículos mais próximos do local do congestionamento têm a prioridade na escolha de uma rota alternativa, veja a Figura 2.7.

Outra aplicação importante dentro de segurança pública é o de detecção de desastre naturais e evacuação. Sabe-se que o grande terremoto que atingiu o leste do Japão[3] e o acidente nuclear de Fukushima[4] (também no Japão) causaram grandes movimentos populacionais e evacuações na região afetada. Logo, compreender e prever tais movimentos e eventos naturais são fundamentais para o planejamento de ajuda humanitária, gestão de desastres e reconstrução social de longo prazo [Zheng et al. 2014].

Seguindo essa ideia, no trabalho de [Song et al. 2013] foi construído um grande banco de dados contendo registros de mobilidade humana coletado do GPS dos *smartphones*. Tal banco possui as mobilidades de mais 1,6 milhões de pessoas que foram realizadas entre os dias de 1º de agosto de 2010 a 31 de julho de 2011. Após o tratamento dos dados foi aplicado um modelo probabilístico que encontrou padrões de comportamentos de evacuação de curto e longo-prazo da população após o desastre. Por meio desse modelo foi possível inferir a mobilidade de evacuação da população de outras cidades japonesas. Em outro trabalho, os autores [Yabe et al. 2016] propuseram um arcabouço para estimar com eficiência os pontos críticos de evacuação após grandes desastres naturais em região urbana usando dados de localização coletados dos *smartphones*.

[3]https://www.livescience.com/39110-japan-2011-earthquake-tsunami-facts.html
[4]https://www.britannica.com/event/Fukushima-accident

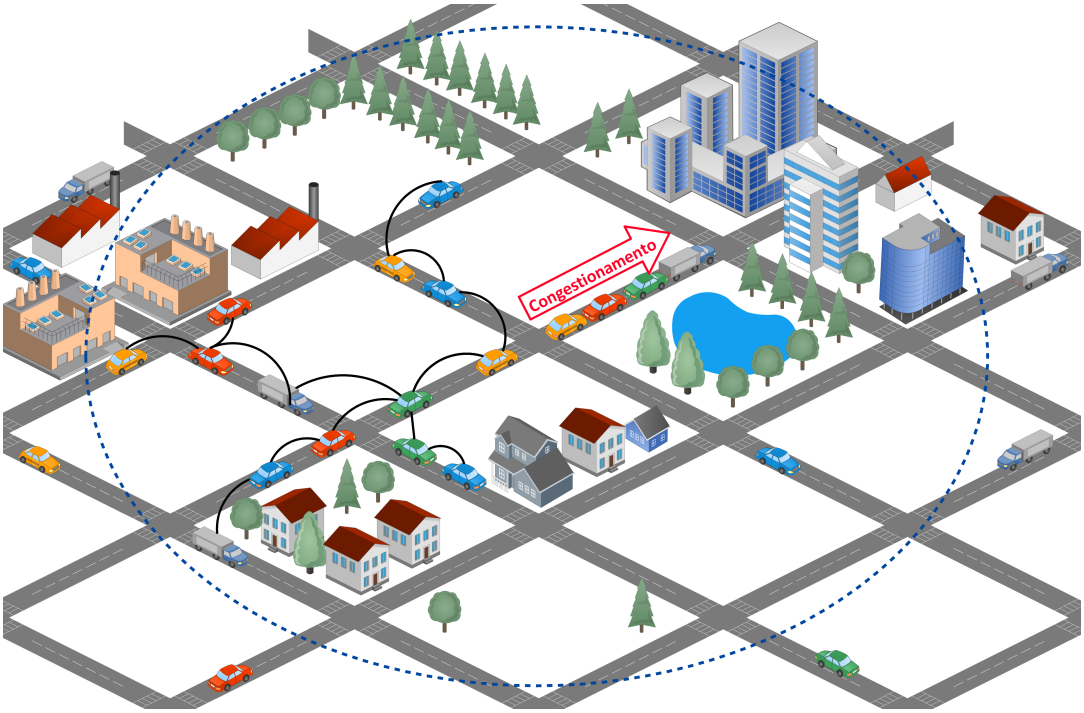


Figura 2.7. Nesse exemplo ilustrativo o veículo mais próximo do congestionamento escolhe primeiro uma rota alternativa e a repassa para a sua vizinhança. O segundo mais próximo utiliza tal rota para calcular a sua e assim por diante. [Akabane et al. 2018a] - modificado.

2.4. Desafios e Oportunidades

Nesta seção serão apresentados os principais desafios enfrentados pela evolução da computação urbana e exploração de novas aplicações para este cenário, destacando oportunidades de pesquisa que podem surgir a partir das ideias apresentadas.

2.4.1. Aquisição e Análise dos Dados

Sob o ponto de vista da aquisição dos dados em ambientes urbanos, diversos desafios podem ser enumerados. Como primeiro deles, a grande variedade dos diferentes modelos de dispositivos. Muitos são os dispositivos e sensores instalados em uma cidade, provenientes de serviços governamentais ou mesmo privados, que geram dados para diferentes propósitos. Além disso, diversos outros aplicativos de *smart phones* são usados por usuários para acessar serviços e coletar dados diariamente. Assim, a integração desses dados provenientes de diferentes plataformas com o objetivo de prover novos serviços para os usuários se torna um grande desafio.

Outro aspecto importante na aquisição dos dados está nas diferenças estruturais presentes nos dados coletados. Questões relativas a granularidade, escalas, precisão dos dados podem dificultar todo o processo de análise e interpretação dos mesmos. Além disso, todo o processo de coleta pode apresentar anomalias e até mesmo algumas falhas, o que pode provocar lacunas no monitoramento realizado. Em situações como essas, o emprego de técnicas estatísticas e modelos matemáticos são fortemente necessários para remover as diferenças entre os dados de forma a obter uma base de dados mais coesa e confiável [Celes et al. 2017].

O aspecto localidade também passar a ser um grande entrave para análise de dados em áreas urbanas. Devido às características de conectividade, concentração de pessoais e de estabelecimentos comerciais, lugares mais populares tendem a ser mais monitorados. Por exemplo, o aplicativo Instagram[5] se torna uma grande fonte de dados acerca do padrão de visitação turística de uma cidade. Entretanto, quando se necessita analisar o padrão de toda a cidade, áreas mais periféricas, afastadas do centro normalmente possuem poucos registros de dados coletados. Nesta circunstância, a correlação de dados de diferentes fontes se torna muito necessária no estudo destas regiões.

Neste mesmo contexto, o armazenamento dos dados também se torna um grande desafio. Alguns dispositivos eletrônicos, devido a suas peculiaridades, apresentam pouca memória, o que demanda o uso de técnicas de compressão e organização dos dados de forma a garantir um melhor gerenciamento de todas as variáveis monitoradas. Assim, algoritmos específicos para trabalhar com os dados coletados e suas diferenças, se tornam essenciais nestes cenários restritivos. Em outra direção, fusão de dados, filtragem e agregação podem ser uma boa estratégia para reduzir a quantidade de dados armazenados e facilitar a análise e indexação dos mesmos [Zheng 2015].

Levando em consideração os dados pessoais, a privacidade passar a ser uma grande barreira para sua coleta e análise. Em muitos cenários, a identificação do dado se torna um fator restritivo para coleta, impedindo que o usuário permita o acesso e o compartilhamento desses dados. Neste contexto, serviços que fazem uso de dados pessoais devem criar mecanismos para dar segurança para seus usuários, incentivando o seu uso. Em outra abordagem, na literatura pode-se encontrar muitos trabalhos que apresentam discussões sobre privacidade e segurança de dados em ambientes urbanos, que apresentam técnicas de anonimização dos dados evitando essa identificação [Ji et al. 2017]. Em contrapartida, novos projetos de leis são propostos pelo governo no intuito de suprir essa lacuna, dando maior proteção aos usuários.

Tendo em vista a análise dos dados urbanos, as diferentes características presentes nas cidades podem dificultar o processo de modelagem e análise dos dados. Cidades apresentam características peculiares, dentre elas pode-se citar: relevo, clima, ocupação do solo, padrões de mobilidade, etc. E a modelagem destes dados se torna uma tarefa não trivial. Em geral, modelos são criados a partir de padrões, e a variação destes padrões tende a inviabilizar a aplicação de um mesmo modelo para representar comportamentos de diferentes ambientes urbanos [Silva 2016]. Assim, novas oportunidades de estudos e definições de modelos que se melhor adéquem a essa diversidade de características se torna essencial no cenário da computação urbana.

2.4.2. Aplicações

Como pode ser observado na Seção 1.3, existem diversas aplicações relacionadas ao âmbito de Computação Urbana. Nada obstante, vários desafios são adicionados devido as exigências de qualidade impostas por cada classe de aplicação, em vista disso será listado os principais desafios relacionado aos tipos abordados na Seção.

Dentro do contexto de Sistemas de ITSs, existem os desafios relacionados a maneira

[5] http://www.instagram.com

eficiente em que estes sistemas irão atuar, visando garantir a segurança dos usuários. A arquitetura dos ITSs possui diversos componentes que garantem a comunicação entre os dispositivos, podendo adotar várias tecnologias, tais como DSRC, Wi-Fi, LTE, Visible Light Communication, satélite [Zheng et al. 2015]. Portanto, pode-se considerar um desafio projetar as comunicações apropriadas neste cenário heterogêneo de tecnologias.

Dados coletados de sistemas de transporte costumava ser dados não pessoais, como dados de fluxo de tráfego e localização de veículos, porém dados pessoais podem ser necessários durante o processamento de algumas aplicações. Nesse contexto, a privacidade é uma das tarefas mais desafiadoras dentro das aplicações de ITS [Zhu et al. 2018]. Os departamentos de transporte devem regulamentar rigorosamente a definição de dados pessoais utilizados nas aplicações, além de fortalecer o gerenciamento da segurança de dados e aplicar algoritmos para melhorar o nível de segurança dos dados coletados.

Os ITS ainda precisam lidar com a dinâmica das cidades, vários dados estão disponíveis para analisar o comportamento das cidades. Dessa forma, a redução de congestionamentos, poluição sonora e ambiental, além de proporcionar uma locomoção mais segura e eficiente são os principais objetivos desse tipo de sistema [Cunha et al. 2017]. Os dados de mobilidade ajudam a assimilar a rotina das pessoas e assim fornecer aplicações que otimizam os meios de transporte.

Outra preocupação recorrente, está relacionada a coleta dos dados, uma vez que os veículos são equipados com uma grande quantidade de sensores. Estes sensores coletam dados que são utilizados para a tomada de decisão dos sistemas de controle de direção. Assim, o principal desafio é extrair as informações desses sensores e obter um conhecimento a partir desses dados, além de relacionar as informações obtidas com dados externos aos veículos.

A predição de tráfego é um dos desafios que influencia no comportamento das aplicações, os motoristas podem solicitar informações sobre a condição das estradas, com o objetivo de evitar possíveis congestionamentos e acidentes. A predição do trafego pode ser realizada com uma cooperação entre veículos, criando uma *Vehicular Fog Computing* (VFC), responsável por sensoriamento, processamento e compartilhamento de informações [Ning et al. 2019].

Além de que existem aplicações gerando uma grande quantidade de dados, encontra-se a necessidade de realizar o *offloading* de maneira eficiente. Uma VFC pode prover esse tipo de serviço buscando reduzir a latência e o custo de disseminação de uma determinada aplicação [Zhang et al. 2017]. Um mecanismo de *offloading* preditivo pode ser projetado, ao estimar o consumo de tempo, para decidir se o envio direto ou o encaminhamento de retransmissão preditiva é adequado para comunicações veiculares. Já no caso de aplicações sociais baseadas em localização a quantidade e a confiabilidade dos dados inseridos pelos usuários são os principais pontos que devem ser examinados. Os dados disponíveis para as aplicações podem representar apenas uma parte dos hábitos dos usuários e a disponibilidade desses dados pode ser afetados por fatores externos, como condições climáticas.

Além disso, não se pode apenas assumir que os dados inseridos pelos usuários estão totalmente corretos, algumas informações podem ser geradas deliberadamente pelos usuários, o que pode introduzir uma inconsistência no que é aferido. Um exemplo disso

pode acontecer quando um determinado usuário realiza um *check-in* falso em um restaurante por estar na moda, o que influenciaria no sistema de recomendação ou popularidade de um determinado local. É importante destacar a etapa de coleta desses dados, uma das dificuldades está no desenvolvimento de ferramentas para coletas, levando em consideração a eficiência, uma vez que se trata de uma grande quantidade de dados provenientes de fontes distintas. Esse mecanismo deve lidar ainda com questões de restrição de privacidade envolvendo os usuários. Além de garantir que os dados coletados sejam verdadeiros[Santos et al. 2017].

Outra dificuldade encontrada nesta área está relacionada as múltiplas fontes de dados urbanos, em aplicações desse tipo é necessário extrair informações de diversas fontes, com dados distintos (Por exemplo, vídeos, imagens e textos). Os algoritmos devem lidar com uma grande quantidade de dados gerados por vários tipos de LBSN, dessa forma, se faz necessário saber como integrar essas fontes heterogêneas de dados, como agregar ou até mesmo desenvolver algoritmos para filtrar tais dados.

A localização desempenha um papel importante nas aplicações citadas, sendo um dos seus principais problemas a ser resolvidos: Como prover dados de posicionamento a qualquer momento e lugar de uma forma mais acurada e confiável? No cenário da computação urbana, a natureza do deslocamento em alta velocidade dos veículos causa mudanças rápidas e constantes na topologia das redes veiculares, o que provoca a disseminação de informações desatualizadas, esse é outro ponto que deve ser levado em consideração quando são propostas aplicações neste sentido.

Aplicações críticas, como as associadas a veículos autônomos, podem exigir dados de localização com baixas taxas de erro. Erros de localização em GPS estão em torno de 10 a 30 metros[Nascimento et al. 2018], o que cria a necessidade de aplicar técnicas para reduzir esses erros, como *Map Matching* e *Dead Reckoning*. Entretanto, ainda é um desafio pertinente prover serviços de localização com uma alta acurácia e em todos os cenários, como tuneis por exemplo[Golestan et al. 2015].

Por fim, nos casos das aplicações relacionadas a segurança urbana, existe a necessidade de levar em consideração os dados disponíveis para propor as rotas mais seguras para os usuários. A maioria dos sistemas propostos foca apenas no menor caminho possível, podendo levar os veículos a lugares perigosos. Nesse contexto é necessário unir os dados de diferentes fontes para achar a melhor rota que combine segurança e tempo.

Outro desafio trata como fazer o cálculo de rotas seguras e eficientes sem adicionar um tempo maior de processamento e evitar congestionamentos. Esse fator depende do número de entidades que serão roteadas (carros, motos, ônibus, etc), o que pode aumentar o tempo necessário para processar todas as informações coletadas e sugerir as melhores rotas[De Souza et al. 2017].

Existem outros desafios relacionados a aplicações específicas e podem ser citados como o processamento em tempo real de grandes quantidades de dados, interação com usuários ou um grupo de usuários, a interação com o ambiente em que estas aplicações estão inseridas, além de uma falta de *testbeds*, o que torna difícil compreender o real comportamento e desafios das aplicações [Santana et al. 2018].

2.5. Um Estudo de Caso: Computação Urbana na Prática

A fim de aumentar a familiaridade da audiência com o arcabouço de computação urbana, apresentamos um estudo de caso que explora a utilização de dados urbanos disponíveis em plataformas online publicamente para identificar fluxos de mobilidade nas cidades, com o intuito de oferecer opções de rotas multimodais para os cidadães. Para isso, utilizamos como cenário a cidade de São Paulo, SP, motivado por diversos fatores, tais como problemas recorrentes de congestionamento que afeta diariamente a vida dos pessoas que moram ou visitam São Paulo e a necessidade das pessoas em percorrer grandes distâncias para deslocarem-se entre suas casas e trabalhos (ou escolas, lugares de lazer, etc), além de haver uma grande quantidade de dados disponíveis que são cruciais para aplicação.

Esta seção está organizada como se segue. A Subseção 2.5.1 apresenta o arcabouço utilizado para identificação de fluxos de mobilidade em São Paulo. Baseado nos fluxos identificados, a Subseção 2.5.2 descreve o passo chave da aplicação, o algoritmo de sugestão de rotas multimodais, que além das rotas multimodais também calcula rotas "tradicionais", i.e., que consideram apenas transporte público ou apenas veículo privado como meio de transporte, para possibilitar a realização de uma análise comparativa entre as rotas sugeridas. Como o custo financeiro para percorrer uma rota com um veículo privado pode variar de acordo com o tipo de veículo, simplificamos ao considerar que é utilizado o serviço de transporte da empresa Uber *Technologies Inc*., assim, os resultados apresentados utilizam a palavra "Uber" para indicar que o meio de transporte considerado é o veículo privado. Também é apresentado um algoritmo para geração de visualização das rotas sugeridas em mapas 2D interativos. A Subseção 2.5.3 demonstra a vantagem em se utilizar as rotas multimodais em comparação as rotas tradicionais ao conseguir um bom equilíbrio em relação as métricas de custo financeiro (preço), tempo de viagem estimado (duração) e distância percorrida a pé. Todos os materiais, algoritmos, ferramentas e tutoriais para instalação e execução da aplicação pode ser acessado no Github[6].

2.5.1. Identificação de Fluxos Urbanos

Como já supracitado, há várias maneiras para extração da mobilidade das pessoas nas cidades (i.e., os fluxos urbanos) a partir de dados urbanos. Entre elas, temos o SMAFramework [Rodrigues et al. 2018], um arcabouço para análise de dados de mobilidade urbana que disponibiliza um algoritmo que permite o agrupamento de viagens urbanas em fluxos de mobilidade e classifica esses fluxos em tendências e secundários. O SMAFramework utiliza o algoritmo HDBSCAN para identificar zonas funcionais nos inícios e finais de viagens da base de dados, utilizadas para computação e classificação de fluxos entre as zonas funcionais. Aqui, o SMAFramework é utilizado como uma "caixa preta" para identificação dos principais fluxos, onde é detalhado o processo para aquisição e processamento da instância de entrada para o algoritmo e a utilização da saída gerada (os fluxos urbanos), mas o funcionamento do SMAFramework é explicado em alto nível. É recomendado que o leitor explore não apenas o SMAFramework, mas também outras abordagens que efetuem a identificação de fluxos urbanos (ou desenvolva sua própria solução), os quais podem ser facilmente adequados para serem utilizados em conjunto com essa aplicação.

[6]https://github.com/diegopso/hybrid-urban-routing-tutorial-sbrc

2.5.1.1. Aquisição de dados

O primeiro passo é coletar uma grande quantidade de dados urbanos que possibilitem a identificação da mobilidade urbana. Como visto anteriormente, as redes sociais baseada em localização (LBSNs) são uma rica fonte de dados geográficos compartilhados pelos usuários em tempo-real. Por isso, é utilizado o Twitter como fonte de dados, que é uma grande LBSN e oferece aos usuários o serviço de *microblogging online*, onde eles podem compartilhar mensagens curtas (máximo de 280 caracteres) conhecidas como *tweets*. Por meio da API do Twitter é possível coletar *tweets* de áreas de interesse, onde parte desse conjunto de *tweets* é geo-localizado. Como a maioria das APIs, a API do twitter também impõem aos seus utilizadores uma série de restrições[7] que devem ser consideradas durante o processo de aquisição de *tweets*.

Um objeto *tweet* [8] coletado por meio da API (renderizado em JSON), é composto por uma longa lista de atributos como o *id* do *tweet*, o conteúdo da mensagem (*text*), informações sobre o usuário (*user*), o tempo em que o *tweet* foi criado em mili-segundos (*timestamp_ms*, onde o fuso horário é UTC), as coordenadas do local se o *tweet* é geolocalizado (*coords*), entre outros. Em geral, o tamanho de um *tweet* renderizado é cerca de 5KB, mas como são coletados centenas de milhares de *tweets*, o quantidade de espaço para armazenamento pode facilmente extrapolar as dezenas e até centenas de GB. Por isso, o conjunto de dados de *tweets* geo-localizados disponibilizado para o minicurso (chamado de `geotagged_tweets_sp.csv`), contém apenas os atributos requeridos pelo SMA-Framework, a saber: um *id* aleatório para o usuário (*uid*), as coordenadas (latitude – *lat* – e longitude – *lon*) e o tempo de criação do *tweet* (*timestamp_ms*), o que reduz drasticamente a quantidade de espaço requerida (de ±3,3GB para ±44MB).

2.5.1.2. Relacionamento e Filtro de Dados

De posse do *dataset* `geotagged_tweets_sp.csv`, o próximo passo consiste em relacionar os *tweets* existentes em fluxos urbanos. Para isso, o primeiro passo é agrupar os *tweets* de acordo com o atributo *uid*, onde cada grupo é composto por *tweets* compartilhados por um mesmo usuário. Então, para cada grupo são criados *links* entre os *tweets*, indicando uma sequência temporal entre os *tweets*. Por exemplo, suponha que o usuário cujo $uid = 1$ tenha compartilhado 10 *tweets* ao longo de um dia, denotado por $tweet^0_{uid=1}, tweet^1_{uid=1}, tweet^2_{uid=1}, ..., tweet^9_{uid=1}$. Assim, um *link* entre dois *tweets*, por exemplo, $tweet^i_{uid=1} \rightarrow tweet^j_{uid=1}$, onde $0 \leq i \neq j < 10$, indica que o *tweet* $tweet^j_{uid=1}$ foi compartilhado subsequente ao *tweet* $tweet^i_{uid=1}$, sendo os *tweets* $tweet^i_{uid=1}$ e $tweet^j_{uid=1}$ contíguos nessa sequência temporal.

Note que ao criar tais *links*, algumas dessas conexões podem não ser interessantes para nossa aplicação. Por exemplo, conexões entre dois *tweets* que foram compartilhados em dias diferentes potencialmente não indica um fluxo válido, uma vez que o usuário pode ter se deslocado para diferentes áreas na janela temporal entre os dois *tweets*, não implicando em fluxo direto entre as localizações em que os *tweets* foram compartilhados.

[7] https://developer.twitter.com/en/docs/basics/rate-limiting.html
[8] https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object.html

Para evitar esse tipo de ruído e alguns outros, foram definidos e utilizados filtros, como apresenta a Tabela 2.1. Essa tabela também mostra para cada passo, a descrição e a quantidade de dados restantes no conjunto de dados, sendo que para o passo Inicial (linha 1), a quantidade é igual o número total de *tweets* e, para as demais linhas, a quantidade é referente ao total de conexões.

	Passo	Descrição	Quantidade
	Inicial	Todos os *tweets* analisados, ainda não agrupados	690139
	Conexão de *tweets*	*Tweets* conectados em sequências temporais	588760
Filtros	Seleção de fluxos diários	Filtro de conexões com datas inicial e final distintas	94411
Filtros	Restrição de distância mínima	Filtro de conexões com distância percorrida menor que 100m	40612
Filtros	Restrição de duração mínima	Filtro de conexões com duração menor que 1 segundo	40567
Filtros	Restrição de velocidade	Filtro de conexões com velocidade menor que 2 Km/h e maior que 100 Km/h	15675

Tabela 2.1. Passos para seleção de fluxos válidos.

Feito isso, a instância de entrada para o algoritmo do SMAFramework está pronta. A partir de todos os fluxos identificados, o SMAFramework determina os principais fluxos de mobilidade, como mostra a Figura 2.8. Ao todo, foram identificados 12 fluxos de mobilidade urbana em São Paulo, SP, a partir dos *tweets* analisados. Como podemos observar, a maioria desses fluxos estão concentrados em quatro regiões, que são os bairros: Jardim Itatinga, Olímpico em São Caetano do Sul, Jardim das Perdizes e Aclimação. Por isso, somente os fluxos entre esses bairros serão considerados para os próximos passos da aplicação (um total de 7 fluxos). Alternativamente, todos os fluxos identificados, além dos 12 resultantes do SMAFramework, poderiam ser avaliados individualmente, contudo, tal análise demandaria muitos recursos computacionais e quantidade de tempo.

2.5.2. Rotas Multimodais

Para percorrer um dado fluxo, podemos considerar três opções de meio de transporte diferentes, são eles: a pé, transporte público (i.e., metrô e ônibus) e veículo privado(nesse caso, Uber). Assim, para cada fluxo de viagem, são calculadas rotas e gerados mapas considerando os seguintes tipos de mobilidade: prioritariamente com transporte público (rotas de cor vermelha), onde alguns trechos da rota possivelmente são realizados a pé (rotas em verde) devido a transição entre as estações; utilizando o serviço de Uber; e as rotas multimodais, onde as três opções de meio de transporte podem ser combinadas de diferentes maneiras. Aqui, apresentamos duas combinações possíveis de rotas multimodais, uma sendo prioritariamente com o transporte público, denotada por "Híbrida 1", e a outra sendo prioritariamente com o Uber, denotada por "Híbrida 2". No entanto, é importante ressaltar que os algoritmos desenvolvidos calculam e geram diversas opções de rotas multimodais. Além disso, como a maioria dos fluxos identificados são demasiadamente longos para serem percorridos totalmente a pé, esse meio de transporte é apenas considerado em conjunto com outros meios de transporte.

Figura 2.8. Principais fluxos identificados.

2.5.2.1. Cálculo de Rotas

Dado os pontos de origem e destino de algum fluxo, o Algoritmo 1 descreve os principais passos para calcular diversas opções de rotas multimodais. Inicialmente, o algoritmo inicializa as seguintes variáveis (linhas 1–4): *driveng_way* que armazena uma rota entre a origem e o destino, considerando o meio de transporte como sendo via veículo privado (carro), e é utilizada para determinar as condições de tráfego no trajeto. Note que várias opções de rota podem ser utilizadas como ponto de partida para avaliar outros trajetos com mesma origem e destino. Aqui, essa rota foi determinada por meio da TomTom Routing API[9]; As listas $transit\_start\_candidates$ e $transit\_end\_candidates$, que vão armazenar os pontos para transição de meios de transporte; A lista *options* que irá armazenar todas as opções de rotas multimodais. É importante ressaltar que o início e o fim da rota também foram adicionados como candidatados à transição (linhas 5–6), isso é feito para obter-se também as rotas tradicionais como opções.

Feito isso, o algoritmo percorre todos as instruções na rota original, i.e., as orientações fornecida aos motoristas para chegar ao seu destino, com o intuito de identificar os trechos congestionados na rota e adicionar seus inícios e finais às listas de pontos de transição (laço das linhas 7–12). Uma vez identificados os pontos de transição, o algoritmo realiza as combinações candidatas de início e fim – ts, te – para compor diferentes rotas multimodais com o apoio das APIs de cálculo de rotas do Google Directions, TomTom Routing e Uber. A API do Google Directions foi usada para avaliar as possíveis rotas, ou trechos, que utilizam transporte público; essa API considera todas as opções de transporte público existente, como metrô, ônibus, tram, etc. A API TomTom Routing é utilizada para calcular a rota referência para selecionar os pontos de transição; ela foi escolhida pois

[9]https://developer.tomtom.com

junto a rota traz informações sobre o congestionamento das vias. Por fim, a API do Uber foi utilizada para estimar o valor e o tempo de espera das rotas que utilizam esse meio de transporte. Todas as opções são concatenadas a lista de opções da rota a fim de serem retornadas como saída do algoritmo (laço das linhas 13–18).

Algoritmo 1: Encontrar opções de rotas multimodais.

Entrada : Origem e destino de um fluxo $(origin, destination)$.
Saída : Lista com as opções de rotas multimodais.

```
1  drive_way ← get_driving_way(origin, destination)
2  transit_start_candidates ← newList()
3  transit_end_candidates ← newList()
4  options ← newList()
5  append(transit_start_candidates, origin)
6  append(transit_end_candidates, destination)
7  foreach (index, step) in drive_way.steps do
8      if is_congested(step) then
9          append(transit_start_candidates, step.origin)
10         append(transit_end_candidates, step.destination)
11     end
12 end
13 foreach ts in transit_start_candidates do
14     foreach te in transit_end_candidates do
15         options ← get_options(origin, ts, te, destination)
16         concat(options, opts)
17     end
18 end
```

2.5.2.2. Visualização de Rotas

Com as opções de rotas calculadas para todos os fluxos estudados, falta gerar os mapas com as rotas sugeridas. O algoritmo 2 mostra os principais passos para geração de mapas 2D interativos. Inicialmente, o algoritmo inicializa o mapa (linha 1), que será gerado como saída com a rota sugerida (informada como entrada do algoritmo). Cada trecho da rota (chamado de *step*) calculado pelo algoritmo 18, possuí uma série de informações, tais como a origem, o destino, o tempo de viagem e, os mais importantes para geração da visualização, a *overview polyline* e o modo de transporte. De acordo com o Google Maps[10], a *overview polyline* é uma representação codificada de um caminho aproximado da rota sugerida e, ao ser decodificada[11], é possível obter uma sequência de pontos (coordenadas) que formam a rota referente ao *step*. Assim, o algoritmo primeiro decodifica a *overview polyline* e armazena em uma lista de coordenadas (linha 3). Então, esta lista de coordenadas é plotada

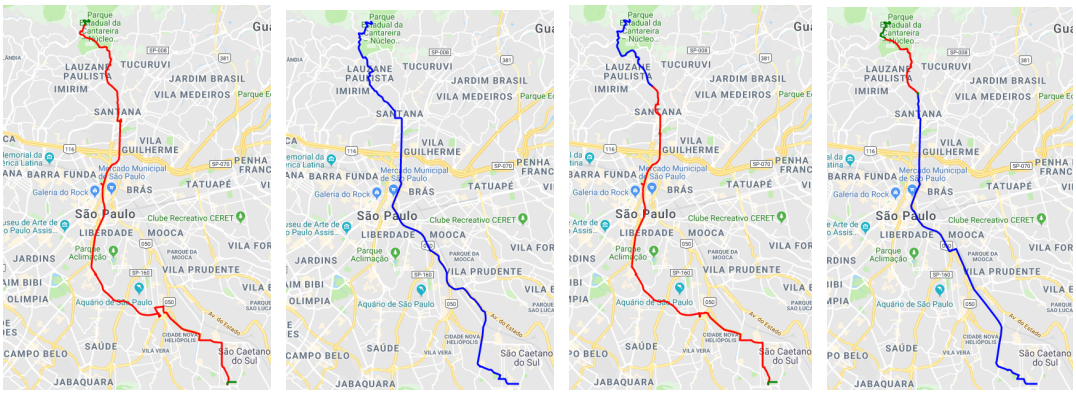
[10] https://developers.google.com/maps/documentation/directions/intro
[11] https://developers.google.com/maps/documentation/utilities/polylinealgorithm

no mapa, onde a cor da linha que representa este caminho é definida de acordo com o meio de transporte e a largura da linha pode ser definida por um valor inteiro (linha 4). Ao final, todos os trechos da rota são plotados no mapa, compondo a rota total com a coloração indicando o meio de transporte em cada trecho. Como resultado, são apresentados a seguir os mapas com as quatro opções de rotas para cada fluxo.

Algoritmo 2: Visualização de rotas.

entrada : *route*.
saída : Mapa com a rota sugerida.

```
// Inicializa o mapa, de acordo com o cenário
   (neste caso, São Paulo).
1 map = setMap(central_coord = (−23.551615, −46.633611), zoom = 12);
2 foreach step ∈ route do
      // Decodifica a overview_polyline
3     path ← decode(step["overview_polyline"]);
      // Adiciona o caminho ao mapa, onde a cor é
         definida pelo meio de transporte.
4     map.plot(path, color = step["travel_mode"], edge_width = 3);
5 end
6 map.draw(output.html)
```

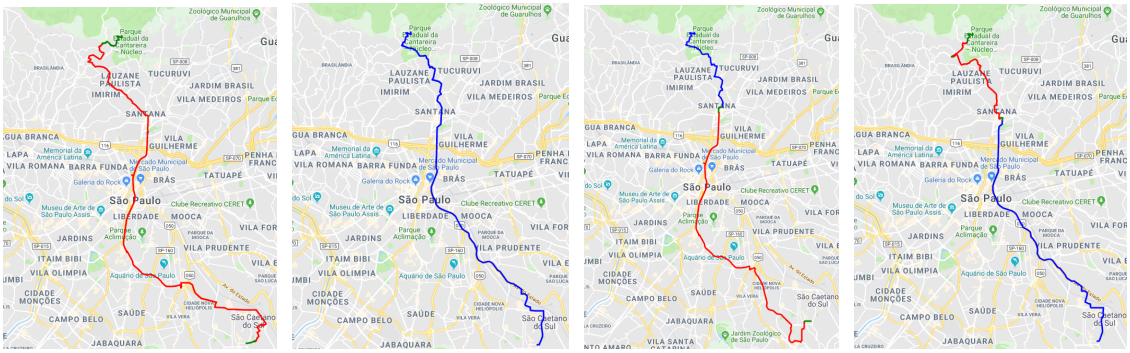


(a) Transporte público. (b) Uber. (c) Híbrida 1. (d) Híbrida 2.

Figura 2.9. Fluxo 1 – Do bairro Jardim Itatinga para o bairro Olímpico, São Caetano do Sul.

2.5.3. Análise Comparativa das Rotas

Considerando as métricas de distância (total e percorrida a pé), preço, tempo estimado de viagem e tempo de espera, realizamos uma análise comparativa das rotas estudadas. Inicialmente, a Figura 2.16 mostra os valores obtidos para essas métricas considerando as rotas apresentadas nas Figuras 2.9–2.15. Com relação a distância (Figura 2.16(a)), os valores obtidos pelas rotas multimodais Híbrida 1 e Híbrida 2 são muito semelhantes as rotas Transporte Público e Uber, respectivamente, para todos os fluxos. Este resultado era esperado, uma vez que ambas as rotas multimodais são compostas em sua maioria por um desses dois tipos de meio de transporte. Diferentemente, as outras três métricas

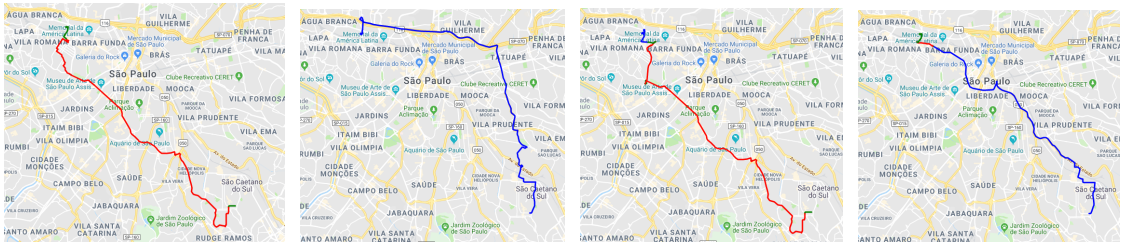


(a) Transporte público. (b) Uber. (c) Híbrida 1. (d) Híbrida 2.

Figura 2.10. Fluxo 2 – Do bairro Olímpico, São Caetano do Sul, para o bairro Jardim Itatinga.

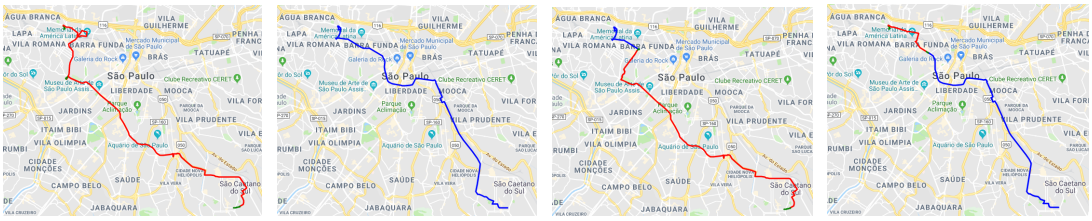
(a) Transporte público. (b) Uber. (c) Híbrida 1. (d) Híbrida 2.

Figura 2.11. Fluxo 3 – Do bairro Jardim Itatinga para o bairro Jardim das Perdizes.



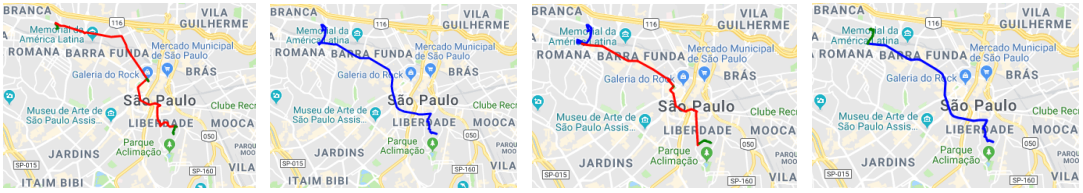
(a) Transporte público. (b) Uber. (c) Híbrida 1. (d) Híbrida 2.

Figura 2.12. Fluxo 4 – Do bairro Olímpico, São Caetano do Sul, para o bairro Jardim das Perdizes.



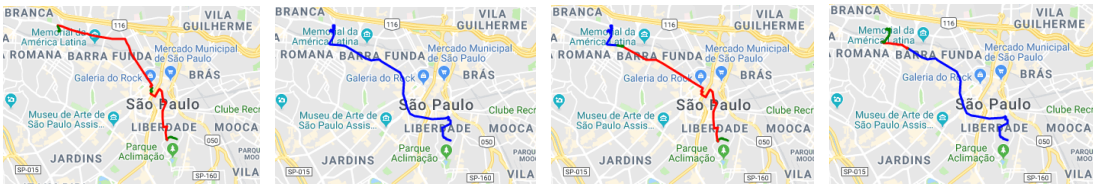
(a) Transporte público. (b) Uber. (c) Híbrida 1. (d) Híbrida 2.

Figura 2.13. Fluxo 5 – Do bairro Jardim das Perdizes para o bairro Olímpico, São Caetano do Sul.



(a) Transporte público. (b) Uber. (c) Híbrida 1. (d) Híbrida 2.

Figura 2.14. Fluxo 6 – Do bairro Jardim das Perdizes para o bairro Aclimação.

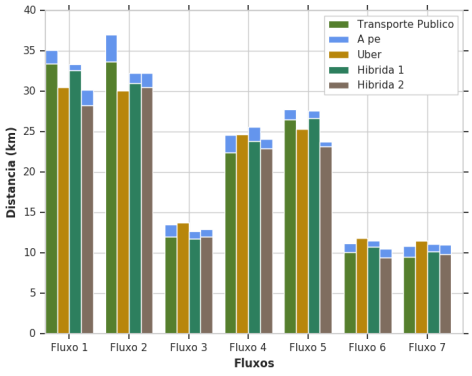


(a) Transporte público. (b) Uber. (c) Híbrida 1. (d) Híbrida 2.

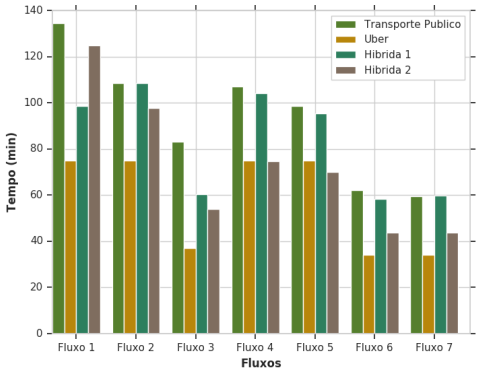
Figura 2.15. Fluxo 7 – Do bairro Aclimação para o bairro Jardim das Perdizes.

apresentam variações significativas na maioria dos fluxos quando são utilizadas rotas multimodais em relação as rotas tradicionais. Por exemplo, o tempo estimado de viagem (Figura 2.16(b)) das rotas Híbrida 2 aumentam consideravelmente em relação as rotas Uber para os fluxos 1, 2, 3, 6 e 7, refletindo o impacto em também ser utilizado o transporte público em conjunto com o Uber, ainda que o modo de transporte Uber seja majoritário nas rotas Híbrida 2. Similarmente, o tempo estimado de viagem das rotas Híbrida 1 diminuem significativamente para os fluxos 1 e 3 em relação as rotas Transporte Público, e, apesar de manter-se similar para os demais fluxos, note que é reduzida a distância percorrida a pé em todos os fluxos quando comparado esses dois tipos de rota. Em relação ao preço (Figura 2.16(c)) e ao tempo de espera (Figura 2.16(d)), é possível observar um relacionamento inverso entre essas métricas, i.e., quanto maior o preço a ser pago, menor é o tempo de espera e vice-versa. É importante ressaltar que foi considerado que o transporte público tem preço R$ 4,30, que é referente a taxa cobrada por trecho na cidade de São Paulo, podendo ser diferente para outros municípios. É possível observar que houveram duas exceções a esse relacionamento, os fluxos 2 e 3, onde a rota Híbrida 1 tem o preço e tempo de espera inferior ao da rota Híbrida 2 em ambos os fluxos. Situações como essa se deve ao momento em que os valores para as métricas foram calculados, onde pode haver uma boa sincronia entre os serviços sem requerer que o usuário aguarde muito entre um meio de transporte e outro. Para evitar casos pontuais como esse, pode-se realizar um experimento empírico, executando o algoritmo 1 em momentos distintos do dia (e.g., manhã, tarde e noite) ao longo de vários dias (e.g., um mês). Assim, é possível obter valores médios para as métricas sem que eventos esporádicos tenham impactos significativos nesse valores.

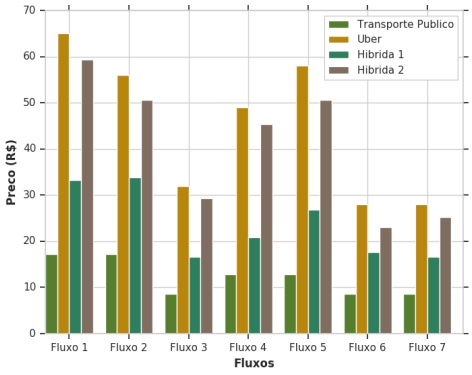
Também é avaliado o impacto médio do uso de rotas multimodais (Híbrida 1 e 2) em comparação com as rotas tradicionais (Transporte Público e Uber) em todos os fluxos estudados, como mostra a Figura 2.17. Como podemos observar, com exceção ao preço, as rotas multimodais são mais viáveis do que as rotas com Transporte Público em todas as demais métricas, onde as principais vantagens são a redução do tempo de viagem médio (cerca de 23 minutos menor) e a redução da distância percorrida a pé. Apesar das rotas com Uber ter a clara vantagem de não ser necessário realizar deslocamentos a pé e, ainda,



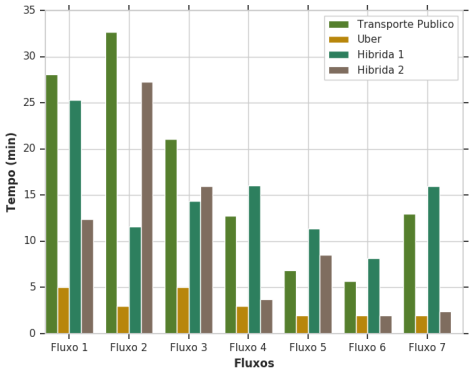
(a) Distância.



(b) Tempo estimado de viagem.

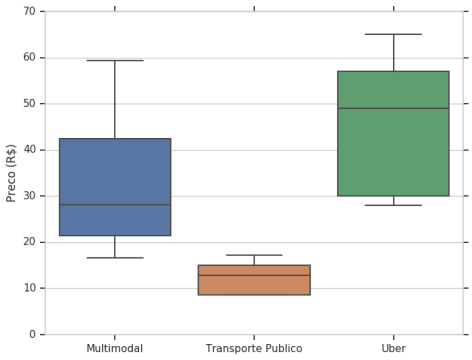


(c) Preço.

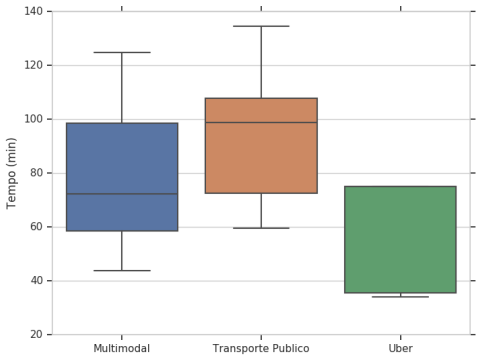


(d) Tempo de espera.

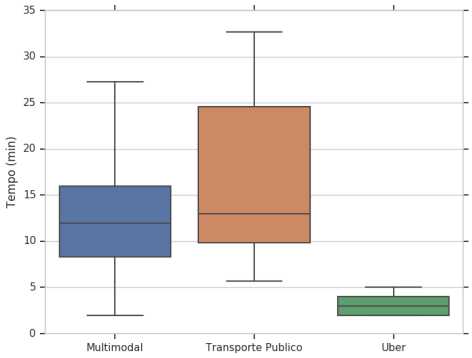
Figura 2.16. Fluxo 3 – Do bairro Jardim Itatinga para o bairro Jardim das Perdizes.



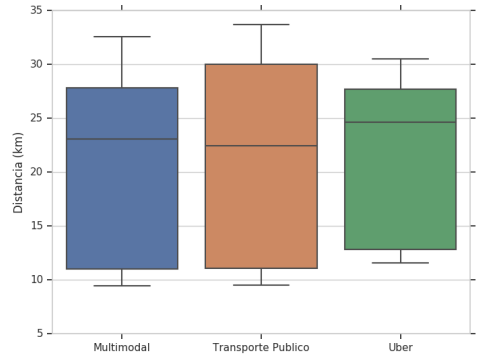
(a) Preço



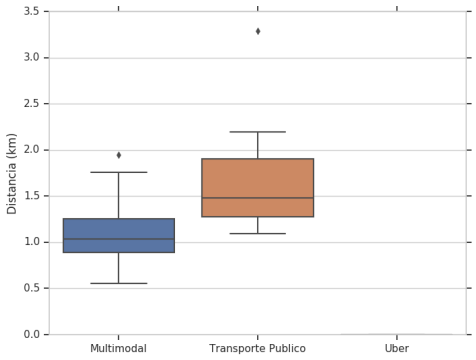
(b) Tempo estimado de viagem.



(c) Tempo de espera.



(d) Distância.



(e) Distância percorrida a pé.

Figura 2.17. Desempenho médio das rotas multimodais e tradicionais, considerando os fluxos de 1 à 7.

ter baixo tempos de espera e de viagem quando comparado com as demais, o preço pode ser um fator crucial na escolha de muitos usuários, tornando esse tipo de rota inviável já que, na média, são cerca de 2 vezes mais caras que as rotas multimodais. Desta forma, pode-se concluir que as rotas multimodais sugeridas pela aplicação alcançam um bom desempenho, sendo mais uma opção para o usuário.

O trabalho *Hybrid Context-Aware Multimodal Routing* [Rodrigues et al. 2018] realiza experimentos similares aos realizados aqui, onde o cenário utilizado é a cidade de Nova Iorque, NY, EUA, e propõem uma função de otimização para calcular a experiência do usuário de acordo com as mesmas métricas apresentadas nesta seção, com o intuito de apoiar no processo de tomada de decisão. Aqui, optamos em apresentar e analisar dois exemplos de rotas multimodais além das rotas tradicionais. Contudo, o único critério para seleção das rotas multimodais foi a composição delas, sendo os dois extremos possíveis (i.e., uma sendo a maior parte com o meio de transporte Uber e a outra com Transporte Público), mas nenhuma das métricas foi utilizada individualmente (e.g., a rota multimodal de menor preço ou tempo de viagem) ou em conjunto (e.g., a rota de menor preço e com menor distância percorrida a pé) para seleção dessas rotas. Caso o leitor tenha a intenção de selecionar as "melhores" rotas multimodais para serem comparadas com as rotas tradicionais, métodos de otimização multiobjetivos devem ser explorados e incorporados à aplicação.

2.6. Conclusão

Devido ao expressivo volume de dados gerados pelos espaços urbanos, bem como nos avanços da tecnologia da informação, estamos diante de uma oportunidade sem precedentes para estudo da Computação Urbana. Diante disso, este minicurso introduziu os fundamentos da Computação Urbana e apresentou os meios necessários para desenvolver uma aplicação nessa área. Para isso, explorou os conceitos que englobam uma visão geral dos trabalhos existentes, os desafios e as perspectivas futuras nessa área em questão. Isso foi útil para apresentar as tendências e ferramentas tipicamente empregadas para o desenvolvimento das aplicações com base na Computação Urbana.

Referências

[Akabane et al. 2018a] Akabane, A. T., Immich, R., Madeira, E. R., and Villas, L. A. (2018a). imob: An intelligent urban mobility management system based on vehicular social networks. In *2018 IEEE Vehicular Networking Conference (VNC)*, pages 1–8. IEEE.

[Akabane et al. 2018b] Akabane, A. T., Immich, R., Pazzi, R. W., Madeira, E. R., and Villas, L. A. (2018b). Trusted: A distributed system for information management and knowledge distribution in vanets. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE.

[Barnaghi et al. 2012] Barnaghi, P., Wang, W., Henson, C., and Taylor, K. (2012). Semantics for the internet of things: Early progress and back to the future. *Int. J. Semant. Web Inf. Syst.*, 8(1):1–21.

[Bellavista and Zanni 2017] Bellavista, P. and Zanni, A. (2017). Feasibility of fog computing deployment based on docker containerization over raspberrypi. In *Proceedings of the 18th International Conference on Distributed Computing and Networking*, ICDCN '17, pages 16:1–16:10, New York, NY, USA. ACM.

[Birari and Iyer 2005] Birari, S. M. and Iyer, S. (2005). Mitigating the reader collision problem in rfid networks with mobile readers. In *2005 13th IEEE International Conference on Networks Jointly held with the 2005 IEEE 7th Malaysia International Conf on Communic*, volume 1, pages 6–pp. IEEE.

[Bittencourt et al. 2018] Bittencourt, L., Immich, R., Sakellariou, R., Fonseca, N., Madeira, E., Curado, M., Villas, L., da Silva, L., Lee, C., and Rana, O. (2018). The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*.

[Bittencourt et al. 2015] Bittencourt, L. F., Lopes, M. M., Petri, I., and Rana, O. F. (2015). Towards virtual machine migration in fog computing. In *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 1–8.

[Byers 2017] Byers, C. C. (2017). Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks. *IEEE Communications Magazine*, 55(8):14–20.

[Celes et al. 2017] Celes, C., Silva, F. A., Boukerche, A., d. C. Andrade, R. M., and Loureiro, A. A. F. (2017). Improving vanet simulation with calibrated vehicular mobility traces. *IEEE Transactions on Mobile Computing*, 16(12):3376–3389.

[Chen et al. 2016] Chen, N., Chen, Y., You, Y., Ling, H., Liang, P., and Zimmermann, R. (2016). Dynamic urban surveillance video stream processing using fog computing. In *2016 IEEE second international conference on multimedia big data (BigMM)*, pages 105–112. IEEE.

[Confais et al. 2017] Confais, B., Lebre, A., and Parrein, B. (2017). *Performance Analysis of Object Store Systems in a Fog and Edge Computing Infrastructure*, pages 40–79. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Cunha et al. 2017] Cunha, F., Maia, G., Celes, C., Guidoni, D., de Souza, F., Ramos, H., and Villas, L. (2017). Sistemas de transporte inteligentes: Conceitos, aplicações desafios. *Livro de Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'17)*.

[Curado et al. 2018] Curado, M., Madeira, H., da Cunha, P. R., Cabral, B., Abreu, D. P., Barata, J., Roque, L., and Immich, R. (2018). *Internet of Things*, pages 381–401. Springer International Publishing.

[De Souza et al. 2017] De Souza, A. M., Brennand, C. A., Yokoyama, R. S., Donato, E. A., Madeira, E. R., and Villas, L. A. (2017). Traffic management systems: A classification, review, challenges, and future perspectives. *International Journal of Distributed Sensor Networks*, 13(4):1550147716683612.

[de Souza et al. 2018] de Souza, A. M., Pedrosa, L. L. C., Botega, L. C., and Villas, L. (2018). Itssafe: An Intelligent Transportation System for Improving Safety and Traffic Efficiency. In *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pages 1–7. IEEE.

[Dib et al. 2015] Dib, O., Manier, M.-A., and Caminada, A. (2015). Memetic algorithm for computing shortest paths in multimodal transportation networks. *Transportation Research Procedia*, 10:745–755.

[Duque et al. 2015] Duque, D., Lozano, L., and Medaglia, A. L. (2015). An exact method for the biobjective shortest path problem for large-scale road networks. *European Journal of Operational Research*, 242(3):788–797.

[Estrin et al. 2010] Estrin, D., Chandy, K. M., Young, R. M., Smarr, L., Odlyzko, A., Clark, D., Reding, V., Ishida, T., Sharma, S., Cerf, V. G., HÃlzle, U., Barroso, L. A., Mulligan, G., Hooke, A., and Elliott, C. (2010). Participatory sensing: applications and architecture [internet predictions]. *IEEE Internet Computing*, 14(1):12–42.

[Filho et al. 2019] Filho, G. P., Villas, L. A., Gonçalves, V. P., Pessin, G., Loureiro, A. A., and Ueyama, J. (2019). Energy-efficient smart home systems: Infrastructure and decision-making process. *Internet of Things*, 5:153 – 167.

[Filho et al. 2013] Filho, G. P. R., Ueyama, J., Villas, L., and Seraphini, A. P. S. (2013). Nodepm: Um sistema de monitoramento remoto do consumo de energia eletrica via redes de sensores sem fio. In sociedade Brasileira de Computacao (SBC), editor, *Simposio Brasileiro de Redes de Computadores e Sistemas Distribuidos (SBRC)*, volume 31, pages 17–30.

[Filho et al. 2018] Filho, G. P. R., Villas, L. A., Freitas, H., Valejo, A., Guidoni, D. L., and Ueyama, J. (2018). Residi: Towards a smarter smart home system for decision-making using wireless sensors and actuators. *Computer Networks*, 135:54 – 69.

[Geraldo Filho et al. 2018] Geraldo Filho, P., Neto, J. R. T., Valejo, A., Meneguette, R. I., Villas, L. A., and Ueyama, J. (2018). Um sistema de controle neuro-fog para infraestruturas residenciais via objetos inteligentes. In *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC.

[Golestan et al. 2015] Golestan, K., Sattar, F., Karray, F., Kamel, M., and Seifzadeh, S. (2015). Localization in vehicular ad hoc networks using data fusion and v2v communication. *Computer Communications*, 71:61–72.

[Group et al. 2017] Group, O. C. A. W. et al. (2017). Openfog reference architecture for fog computing. *OPFRA001*, 20817:162.

[Guo et al. 2014] Guo, C., Jensen, C. S., and Yang, B. (2014). Towards total traffic awareness. *SIGMOD Record*, 43(3):18–23.

[Heintz et al. 2016] Heintz, B., Chandra, A., Sitaraman, R. K., and Weissman, J. (2016). End-to-end optimization for geo-distributed mapreduce. *IEEE Transactions on Cloud Computing*, 4(3):293–306.

[Hung et al. 2015] Hung, C.-C., Golubchik, L., and Yu, M. (2015). Scheduling jobs across geo-distributed datacenters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, pages 111–124, New York, NY, USA. ACM.

[Idri et al. 2017] Idri, A., Oukarfi, M., Boulmakoul, A., Zeitouni, K., and Masri, A. (2017). A new time-dependent shortest path algorithm for multimodal transportation network. *Procedia Computer Science*, 109:692–697.

[Issarny et al. 2018] Issarny, V., Bouloukakis, G., Georgantas, N., Sailhan, F., and Texier, G. (2018). When service-oriented computing meets the iot: A use case in the context of urban mobile crowdsensing. In *European Conference on Service-Oriented and Cloud Computing*, pages 1–16. Springer.

[Ji et al. 2017] Ji, S., Mittal, P., and Beyah, R. (2017). Graph data anonymization, de-anonymization attacks, and de-anonymizability quantification: A survey. *IEEE Communications Surveys Tutorials*, 19(2):1305–1326.

[Kim 2015] Kim, S. (2015). *How Foursquare and Other Apps Guess What You Want to Eat*. Eater.com. https://goo.gl/icFBrH.

[Krasnogor and Smith 2005] Krasnogor, N. and Smith, J. (2005). A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488.

[Meneguette et al. 2016a] Meneguette, R., Fillho, G., Bittencourt, L., Ueyama, J., and Villas, L. (2016a). A solution for detection and control for congested roads using vehicular networks. *IEEE Latin America Transactions*, 14(4):1849–1855.

[Meneguette et al. 2016b] Meneguette, R. I., Geraldo Filho, P., Guidoni, D. L., Pessin, G., Villas, L. A., and Ueyama, J. (2016b). Increasing intelligence in inter-vehicle communications to reduce traffic congestions: experiments in urban and highway environments. *PLoS one*, 11(8):e0159110.

[Moura et al. 2018] Moura, D. L., Cabral, R. S., Sales, T., and Aquino, A. L. (2018). An evolutionary algorithm for roadside unit deployment with betweenness centrality preprocessing. *Future Generation Computer Systems*, 88:776 – 784.

[Mueller et al. 2017] Mueller, W., Silva, T. H., Almeida, J. M., and Loureiro, A. A. (2017). Gender matters! analyzing global cultural gender preferences for venues using social sensing. *EPJ Data Science*, 6(1):5.

[Naboulsi and Fiore 2013] Naboulsi, D. and Fiore, M. (2013). On the instantaneous topology of a large-scale urban vehicular network: the cologne case. In *Proceedings of the fourteenth ACM international symposium on Mobile ad hoc networking and computing*, pages 167–176. ACM.

[Nascimento et al. 2018] Nascimento, P., Kimura, B., Guidoni, D., and Villas, L. (2018). An integrated dead reckoning with cooperative positioning solution to assist gps nlos using vehicular communications. *Sensors*, 18(9):2895.

[Ning et al. 2019] Ning, Z., Huang, J., and Wang, X. (2019). Vehicular fog computing: Enabling real-time traffic management for smart cities. *IEEE Wireless Communications*, 26(1):87–93.

[Pereira Rocha Filho et al. 2018] Pereira Rocha Filho, G., Yukio Mano, L., Demetrius Baria Valejo, A., Aparecido Villas, L., and Ueyama, J. (2018). A low-cost smart home automation to enhance decision-making based on fog computing and computational intelligence. *IEEE Latin America Transactions*, 16(1):186–191.

[Ribeiro et al. 2014] Ribeiro, A. I. J. a. T., Silva, T. H., Duarte-Figueiredo, F., and Loureiro, A. A. (2014). Studying traffic conditions by analyzing foursquare and instagram data. In *Proceedings of the 11th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*, PE-WASUN '14, pages 17–24, New York, NY, USA. ACM.

[Riveiro et al. 2017] Riveiro, M., Lebram, M., and Elmer, M. (2017). Anomaly detection for road traffic: A visual analytics framework. *IEEE Transactions on Intelligent Transportation Systems*, 18(8):2260–2270.

[Rodrigues et al. 2018] Rodrigues, D. O., Boukerche, A., Silva, T. H., Loureiro, A. A., and Villas, L. A. (2018). Combining taxi and social media data to explore urban mobility issues. *Computer Communications*, 132:111 – 125.

[Rodrigues et al. 2018] Rodrigues, D. O., Fernandes, J. T., Curado, M., and Villas, L. A. (2018). Hybrid context-aware multimodal routing. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2250–2255.

[S. Gama et al. 2018] S. Gama, E., Immich, R., and F. Bittencourt, L. (2018). Towards a multi-tier fog/cloud architecture for video streaming. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 13–14.

[Sakr et al. 2011] Sakr, S., Liu, A., Batista, D. M., and Alomari, M. (2011). A survey of large scale data management approaches in cloud environments. *IEEE Communications Surveys Tutorials*, 13(3):311–336.

[Santana et al. 2018] Santana, E. F. Z., Chaves, A. P., Gerosa, M. A., Kon, F., and Milojicic, D. S. (2018). Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. *ACM Computing Surveys (CSUR)*, 50(6):78.

[Santos et al. 2018] Santos, F. A., Silva, T. H., , Loureiro, A. A. F., and Villas, L. A. (2018). Uncovering the Perception of Urban Outdoor Areas Expressed in Social Media. In *Proc. of IEEE ACM International Conference on Web Intelligence (WI)*, Santiago, Chile.

[Santos et al. 2017] Santos, F. A., Silva, T. H., Braun, T., Loureiro, A. A., and Villas, L. A. (2017). Towards a sustainable people-centric sensing. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

[Sedeno-Noda and Raith 2015] Sedeno-Noda, A. and Raith, A. (2015). A dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem. *Computers & Operations Research*, 57:83–94.

[Shi et al. 2016] Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646.

[Silva 2016] Silva, T H., d. M. P. N. J. I. A. d. R. C. M. V. d. C. F. F. A. A. J. L. A. (2016). Users in the urban sensing process: Challenges and research opportunities. In *Pervasive Computing: Next Generation Platforms for Intelligent Data Collection*, pages 45–95. Academic Press.

[Silva et al. 2018] Silva, T., Viana, A., Benevenuto, F., Villas, L., Salles, J., Loureiro, A., and Quercia, D. (2018). Urban Computing Leveraging Location-Based Social Network Data: a Survey. *ACM Computing Surveys*.

[Silva et al. 2014] Silva, T. H., Vaz de Melo, P. O. S., Almeida, J. M., Salles, J., and Loureiro, A. A. F. (2014). Revealing the city that we cannot see. *ACM Trans. Internet Technol.*, 14(4):26:1–26:23.

[Silva et al. 2013] Silva, T. H., Vaz de Melo, P. O. S., Viana, A., Almeida, J. M., Salles, J., and Loureiro, A. A. F. (2013). Traffic Condition is more than Colored Lines on a Map: Characterization of Waze Alerts. In *Proc. of the International Conference on Social Informatics (SocInfo'13)*, Kyoto, Japan.

[Song et al. 2010] Song, C., Qu, Z., Blumm, N., and Barabási, A.-L. (2010). Limits of predictability in human mobility. *Science*, 327(5968):1018–1021.

[Song et al. 2013] Song, X., Zhang, Q., Sekimoto, Y., Horanont, T., Ueyama, S., and Shibasaki, R. (2013). Modeling and probabilistic reasoning of population evacuation during large-scale disaster. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239. ACM.

[SteadieSeifi et al. 2014] SteadieSeifi, M., Dellaert, N. P., Nuijten, W., Van Woensel, T., and Raoufi, R. (2014). Multimodal freight transportation planning: A literature review. *European journal of operational research*, 233(1):1–15.

[Taleb et al. 2017a] Taleb, T., Dutta, S., Ksentini, A., Iqbal, M., and Flinck, H. (2017a). Mobile edge computing potential in making cities smarter. *Comm. Mag.*, 55(3):38–43.

[Taleb et al. 2017b] Taleb, T., Ksentini, A., and Frangoudis, P. (2017b). Follow-me cloud: When cloud services follow mobile users. *IEEE Transactions on Cloud Computing*, pages 1–1.

[Traynor and Curran 2012] Traynor, D. and Curran, K. (2012). Location-based social networks. *From Government to E-Governance: Public Administration in the Digital Age*, page 243.

[United Nations and Affairs 2018] United Nations, D. o. E. and Affairs, S. (2018). *World Urbanization Prospects: The 2018 Revision, Highlights*. United Nations.

[Vaquero and Rodero-Merino 2014] Vaquero, L. M. and Rodero-Merino, L. (2014). Finding your way in the fog: Towards a comprehensive definition of fog computing. *SIGCOMM Comput. Commun. Rev.*, 44(5):27–32.

[Wang et al. 2017] Wang, H., Wen, H., Yi, F., Zhu, H., and Sun, L. (2017). Road traffic anomaly detection via collaborative path inference from gps snippets. *Sensors*, 17(3):550.

[Wang et al. 2018] Wang, X., Ning, Z., Hu, X., Ngai, E. C.-H., Wang, L., Hu, B., and Kwok, R. Y. (2018). A city-wide real-time traffic management system: Enabling crowdsensing in social internet of vehicles. *IEEE Communications Magazine*, 56(9):19–25.

[Wen et al. 2017] Wen, Z., Yang, R., Garraghan, P., Lin, T., Xu, J., and Rovatsos, M. (2017). Fog orchestration for internet of things services. *IEEE Internet Computing*, 21(2):16–24.

[Xu and Li 2013] Xu, H. and Li, B. (2013). Joint request mapping and response routing for geo-distributed cloud services. In *2013 Proceedings IEEE INFOCOM*, pages 854–862.

[Xu et al. 2014] Xu, L. D., He, W., and Li, S. (2014). Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243.

[Yabe et al. 2016] Yabe, T., Tsubouchi, K., Sudo, A., and Sekimoto, Y. (2016). A framework for evacuation hotspot detection after large scale disasters using location data from smartphones: case study of kumamoto earthquake. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 44. ACM.

[Yang et al. 2017] Yang, C., Huang, Q., Li, Z., Liu, K., and Hu, F. (2017). Big data and cloud computing: innovation opportunities and challenges. *International Journal of Digital Earth*, 10(1):13–53.

[Yi et al. 2015] Yi, S., Hao, Z., Qin, Z., and Li, Q. (2015). Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78. IEEE.

[Zhang et al. 2017] Zhang, K., Mao, Y., Leng, S., He, Y., and Zhang, Y. (2017). Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading. *IEEE Vehicular Technology Magazine*, 12(2):36–44.

[Zheng et al. 2015] Zheng, K., Zheng, Q., Chatzimisios, P., Xiang, W., and Zhou, Y. (2015). Heterogeneous vehicular networking: a survey on architecture, challenges, and solutions. *IEEE communications surveys & tutorials*, 17(4):2377–2396.

[Zheng 2011] Zheng, Y. (2011). *Location-Based Social Networks: Users*, pages 243–276. Springer New York, New York, NY.

[Zheng 2012] Zheng, Y. (2012). Tutorial on Location-Based Social Networks. In *Proc. of WWW'12*, Lyon, France.

[Zheng 2015] Zheng, Y. (2015). Methodologies for cross-domain data fusion: An overview. *IEEE Transactions on Big Data*, 1(1):16–34.

[Zheng et al. 2014] Zheng, Y., Capra, L., Wolfson, O., and Yang, H. (2014). Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):38.

[Zhu et al. 2018] Zhu, L., Yu, F. R., Wang, Y., Ning, B., and Tang, T. (2018). Big data analytics in intelligent transportation systems: a survey. *IEEE Transactions on Intelligent Transportation Systems*, 20(1):1–16.

[Zografos and Androutsopoulos 2008] Zografos, K. G. and Androutsopoulos, K. N. (2008). Algorithms for itinerary planning in multimodal transportation networks. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):175–184.

Capítulo 3

Processamento Rápido de Pacotes com eBPF e XDP

Marcos A. M. Vieira, Racyus D. G. Pacífico, Matheus S. Castanho, Elerson R. S. Santos, Eduardo P. M. Câmara Júnior, Luiz F. M. Vieira

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, MG, Brasil
{mmvieira,racyus,matheus.castanho,elerson,epmcj,lfvieira}@dcc.ufmg.br

Abstract

Extended Berkeley Packet Filter (eBPF) has been rapidly adopted across multiple systems since its introduction in the Linux kernel in 2014. eBPF is used for fast packet processing. The use cases of eBPF have grown rapidly to include network monitoring, network traffic manipulation, load balancing, system monitoring, and so on. Several companies already use eBPF on projects such as Facebook, Netronome, and Cilium. This short-course aims to present eBPF. eBPF allows programming network devices. The developer can write in P4 or C language and then compile for eBPF instructions. The eBPF code can be processed in the Linux kernel or by programmable devices such as NetFPGAs and smartNICs. This short-course covers the main theoretical and fundamental aspects of eBPF, as well as introducing the reader to simple practical activities that can give insight into the general operation and use of eBPF.

Resumo

O filtro de pacotes estendido (Extended Berkeley Packet Filter (eBPF)) foi rapidamente adotado em vários sistemas desde sua introdução no kernel do Linux em 2014. O eBPF é utilizado para processamento rápido de pacotes. Os usos do eBPF cresceram rapidamente para incluir monitoramento de rede, manipulação de tráfego de rede, balanceamento de carga, monitoramento do sistema, etc. Várias empresas já utilizam eBPF em projetos como o Facebook, Netronome, Cilium. Este minicurso tem como objetivo apresentar o eBPF. O eBPF permite a programação dos dispositivos de redes. O desenvolvedor pode escrever em linguagem P4 ou C e depois compilar para instruções eBPF.

Depois, o código eBPF pode ser processado no kernel do Linux ou por dispositivos programáveis como NetFPGAs e smartNICs. O minicurso cobre os principais aspectos teóricos e fundamentais do eBPF, assim como introduzir o leitor a atividades práticas simples que possam dar uma visão sobre o funcionamento e uso geral do eBPF.

3.1. Introdução e Motivação

O constante aumento do tráfego na Internet e o crescimento da complexidade de serviços oferecidos em redes de centros de dados têm exigido taxas de processamento de pacotes cada vez mais altas. Além disso, a dinamicidade das demandas de serviços requer que a rede se adapte rapidamente para manter níveis adequados de qualidade de serviço e utilizar recursos disponíveis de forma eficiente. Porém, redes de computadores têm sido tradicionalmente desenvolvidas de forma estática, embutindo a implementação de protocolos de comunicação no *hardware* de dispositivos de rede, dificultando sua adequação às demandas atuais.

Nos últimos anos diversas propostas têm surgido visando adicionar maior programabilidade à rede. Dentre elas os paradigmas de SDN [Feamster et al. 2014] e NFV [Mijumbi et al. 2016], as linguagens POF [Song 2013] e P4 [Bosshart et al. 2014], e mais recentemente o filtro de pacotes estendido (*extended Berkeley Packet Filter* - eBPF) e o caminho de dados expresso (*eXpress Data Path* - XDP).

Este minicurso tem como objetivo apresentar o eBPF e o XDP para a comunidade brasileira de pesquisadores em redes de computadores. O eBPF é utilizado para modificar o processamento de pacotes no *kernel* do Linux e permite também a programação de dispositivos de rede. O desenvolvedor pode escrever uma aplicação na linguagem C ou P4 e depois compilá-la para instruções eBPF. O código eBPF resultante pode ser processado no *kernel* do Linux ou por dispositivos programáveis como NetFPGAs e placas de interface de rede inteligentes (*SmartNICs*).

O XDP fornece um caminho de dados de rede programável de alto desempenho no *kernel* do Linux. Ele efetua processamento de pacotes no ponto mais baixo da pilha de software, o que o torna ideal para aplicações de alta velocidade sem comprometer a programabilidade. Além disso, novas funções podem ser implementadas dinamicamente com o caminho rápido integrado sem modificação do código fonte do *kernel*. O programa eBPF modifica o funcionamento do *kernel* (programável), porém não precisa recompilá-lo pra isso.

A importância do eBPF e XDP é percebida pela sua rápida adoção desde sua introdução no *kernel* do Linux em 2014, tanto pela indústria e quanto por projetos de pesquisa na academia. Seus casos de uso cresceram rapidamente para incluir tarefas como monitoramento de rede, manipulação de tráfego de rede, balanceamento de carga e introspecção do sistema operacional. Várias empresas já utilizam eBPF em projetos, como Facebook [Facebook 2018], Netronome [Beckett et al. 2018] e Cloudflare [Bertin 2017].

O minicurso está organizado da seguinte forma: o restante desta seção apresenta uma introdução ao BPF. Na seção 3.2, é descrita a arquitetura da máquina eBPF. A seção 3.3 explica a visão geral do sistema eBPF. Na seção 3.4, é definido o que são ganchos (*hooks*) e onde eles ocorrem na pilha de rede. Também é explicado o que é XDP e como

utilizá-lo. A seção 3.5 descreve aspectos dos programas eBPFs, como os tipos de programas disponíveis, o que são mapas, os tipos de mapas disponíveis e como utilizá-los em um programa eBPF. Também define-se o que são funções auxiliares e tem-se a descrição das presentes no *kernel* do Linux. Na seção 3.6 são apresentadas algumas ferramentas úteis para o desenvolvimento e depuração de programas eBPF. A seção 3.7 descreve as plataformas de software e hardware já existentes capazes de processar instruções eBPF. Na seção 3.8 são apresentadas exemplos de funções de rede que utilizam o eBPF. A seção 3.9 discute vários projetos de pesquisas importantes já desenvolvidos e a seção 3.10 apresenta as atuais limitações da arquitetura e dá sugestões de como superá-las. Finalmente, a seção 3.11, conclui o minicurso. Todo o código utilizado neste minicurso está disponível no [Vieira et al. 2019].

3.1.1. O que é BPF

O filtro de pacotes BPF (*Berkeley Packet Filter*) [McCanne and Jacobson 1993] foi proposto por Steven McCanne and Van Jacobson em 1992, como uma solução para realizar filtragem de pacotes no *kernel* de sistemas Unix BSD. Ele consistia em um conjunto de instruções e uma máquina virtual (VM) para execução de programas escritos nessa linguagem.

Inicialmente, o código de bytes (bytecode) de uma aplicação era transferido do espaço de usuário para o *kernel*, onde era então verificado para prevenir problemas de segurança e falhas no *kernel*. Após passar na verificação, o programa era anexado a um soquete e executado a cada pacote recebido. Essa habilidade de executar programas fornecidos pelo usuário no *kernel* de forma segura se mostrou uma boa escolha de design do BPF.

Outro fator de destaque do BPF é seu conjunto de instruções simples e bem definido. Aliado à existência de uma máquina de compilação JIT (*Just-In-Time*) para BPF no *kernel*, esses fatores foram fundamentais para o bom desempenho da ferramenta.

Além da definição do código de bytes, o BPF também define um modelo de memória baseado em pacotes (instruções de carga são implicitamente feitas no pacote de processamento), registradores (A e X, acumulador e registrador de índice), um armazenamento de memória temporário e um contador de programa implícito. O lado esquerdo da figura 3.1 (Máquina BPF Clássica) ilustra a arquitetura da máquina BPF.

O *kernel* do Linux possui suporte ao BPF desde a versão 2.5. Não houve grandes alterações no código BPF até 2011, quando o interpretador BPF foi modificado para ser um tradutor dinâmico [Dumazet 2011]. Em vez de interpretar o código de bytes BPF, agora o *kernel* era capaz de traduzir os programas BPF diretamente para uma arquitetura de destino: x86, ARM, MIPS, etc.

Um dos usuários mais proeminentes do BPF é a biblioteca *libpcap*, presente na ferramenta *tcpdump*. Ao utilizar o *tcpdump* para capturar pacotes, um usuário pode definir uma expressão de filtragem de pacotes de modo que somente pacotes que correspondem a essa expressão serão realmente capturados. Por exemplo, a expressão "*ipv4 tcp*" captura todos os pacotes IPv4 que contem o protocolo da camada de transporte TCP. Essa expressão pode ser reduzida por um compilador para o código de bytes BPF. O código 1

apresenta um exemplo de programa BPF para filtrar pacotes permitindo capturar apenas segmentos TCP.

Código 1 Exemplo de program BPF para permitir apenas segmentos TCP.

```
(000) ldh [12]
(001) jeq #0x800 jt 2 jf 5
(002) ldb [23]
(003) jeq #6  jt 4 jf 5
(004) ret #-1
(005) ret #0
```

Basicamente, o que o código 1 faz é:

- Instrução (000): carrega (*load (ld)*) o *offset* 12 do quadro, como uma palavra de 16 bits, no acumulador. O deslocamento 12 representa o tipo de pacote no quadro Ethernet.
- Instrução (001): compara o valor do acumulador com 0x800, que é o valor *EtherType* do IPv4. Se o resultado for verdadeiro, o contador do programa salta (*jt jump true - desvio se verdadeiro*) para a instrução (002), caso contrário salta *jf jump falso - desvio se falso* para a instrução (005).
- Instrução (002): carrega o *offset* 23 do quadro, como um byte, no acumulador. O deslocamento 23 representa o campo protocolo do pacote IPv4. A contagem é a partir do início do quadro Ethernet.
- Instrução (003): compara o valor com 6 (valor do campo protocolo do pacote IPv4 que indica se contem um segmento TCP). Se for verdadeiro, pular para a instrução (004), se não ir para a instrução (005).

O programa de filtragem de pacotes executa até retornar um resultado, que geralmente é um booleano. Retornar um valor diferente de zero (instrução (004)) significa que o pacote casou com o filtro, enquanto que retornar um valor zero (instrução (005)) significa que o pacote não corresponde com o filtro e por isso será descartado.

3.2. Máquina eBPF

Ao longo do tempo, alguns aspectos de design do BPF não conseguiram se sustentar tão bem. O design da VM e da arquitetura do conjunto de instruções (ISA) ficaram defasados na medida que os processadores modernos migraram para registradores de 64 bits e novas instruções necessárias para sistemas multiprocessadores foram adicionadas. Com isso, o foco do BPF em prover um número pequeno de instruções RISC deixou de corresponder às realidades dos processadores modernos.

Dessa forma, uma nova versão BPF foi introduzida para aproveitar os novos recursos dos hardwares modernos. Essa nova versão foi batizada de eBPF, enquanto a anterior se tornou o cBPF (BPF clássico) e foi introduzida na versão 3.15 do *kernel*.

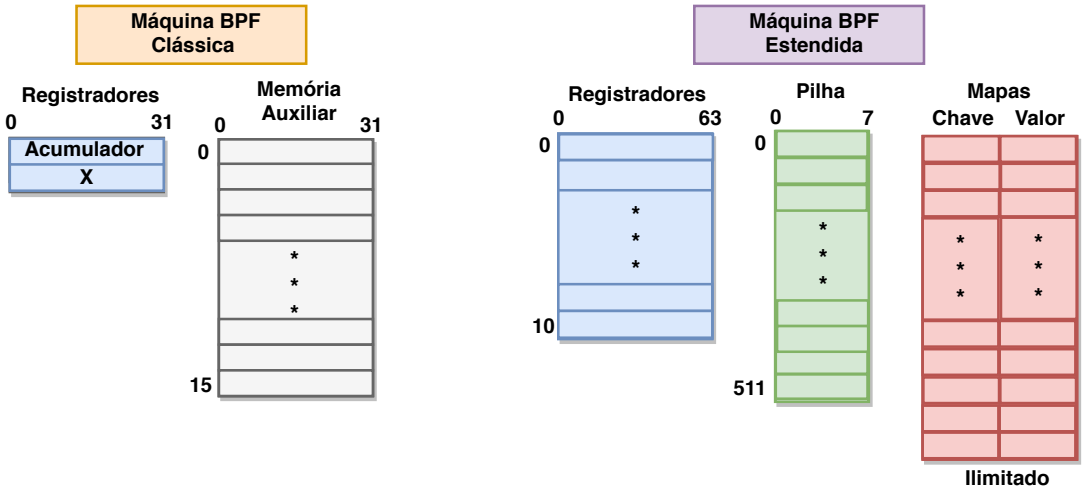


Figura 3.1. Processador BPF e eBPF.

O lado direito da figura 3.1 ilustra a máquina eBPF. O número de registradores aumentou de 2 para 11 (dos quais 10 registradores podem ser escritos), a largura dos registradores passou de 32 bits para 64 bits, o conjunto de instruções passou a ser de 64 bits e a máquina agora possui uma pilha de 512 bytes. Também foram adicionados armazenamentos de dados globais, denominados mapas, e a opção de chamadas de funções que são executadas dentro do *kernel*, denominadas funções auxiliares [Schulist et al. 2019].

O eBPF também adicionou o recurso de chamadas de cauda (*tail-calls*) para contornar a limitação de tamanho máximo de 4096 bytes dos programas eBPF. Com ele, um programa eBPF é capaz de passar o controle de execução para um novo programa eBPF.

No cBPF era necessário definir os desvios para os casos verdadeiros e falsos em um programa. Já no eBPF só é necessário definir os desvios verdadeiros, sendo que os desvios falsos seguem a sequência de execução do programa (chamados *jump fall-through*).

O eBPF também adicionou novas instruções ao conjunto de instruções. Entre as inclusões estão instruções aritméticas e lógicas para os registradores, bem como uma instrução para chamadas de função no *kernel* de forma barata. O eBPF adota a mesma convenção de chamada da linguagem C. Parâmetros são passados para funções através de registradores da máquina virtual, assim como acontece nativamente no hardware. Isso permite mapear uma instrução de chamada de função eBPF para uma única instrução de chamada nativa, habilitando a chamada da função quase sem custo. Esta facilidade é utilizada pelo eBPF para suportar as funções auxiliares, permitindo que os programas eBPF interajam com o *kernel* durante o processamento e realizem chamadas de sistema.

A máquina virtual eBPF suporta o carregamento dinâmico e o recarregamento de programas. O *kernel* gerencia o ciclo de vida de todos os programas. Isso permite estender ou limitar a quantidade de processamento realizado para uma determinada situação, adicionando ou removendo partes do programa que não são necessárias e recarregá-las novamente.

3.2.1. Classes de instruções e conjunto de instruções

O eBPF possui sete classes de instruções: *load* imediato (BPF_LD), *load* estendido para 64 bits (BPF_LDX), *store* imediato (BPF_ST), *store* estendido para 64 bits (BPF_STX), operações lógicas e aritméticas (BPF_ALU), desvios (BPF_JMP), operações lógicas e aritméticas de 64 bits (BPF_ALU64).

BPF_LD, BPF_LDX: Ambas as classes são para operações de carregamento. O BPF_LD é usado para carregar uma palavra dupla como uma instrução especial utilizando duas instruções devido ao imediato ter apenas 32 bits e precisar de 64 bits. Também é usado para carregar 8 bits (byte (B)), 16 bits (meia palavra *half word*(H)), 32 bits (palavra *word*(w)), e 64 bits (palavra dupla *double word* (DW)). A classe BPF_LDX contém instruções para carregamentos de byte / meia palavra / palavra / palavra dupla fora da memória. A memória neste contexto é genérica e pode ser memória de pilha, dados de valor de mapa, dados de pacote, etc.

BPF_ST, BPF_STX: Ambas as classes são para operações de armazenamento. Semelhante a BPF_LDX, o BPF_STX é a extensão de BPF_ST e é usado para armazenar os dados de um registrador na memória, que, novamente, pode ser da pilha, valor de mapa, dados de pacote, etc. BPF_STX também contém instruções especiais para executar palavras e palavras duplas baseados em operações de adição atômica, que podem ser usadas para contadores, por exemplo. A classe BPF_ST é semelhante a BPF_STX fornecendo instruções para armazenar dados na memória apenas que o operando de origem é um valor imediato.

BPF_ALU, BPF_ALU64: Ambas as classes contêm operações da ULA (Unidade Lógica e Aritmética). Geralmente, as operações BPF_ALU estão no modo de 32 bits e BPF_ALU64 no modo de 64 bits. Ambas as classes da ULA possuem operações básicas com o operando de origem, que é baseado em registro e uma contraparte baseada em imediato. As operações suportadas por ambos são: soma, subtração, e lógico, ou lógico, deslocamento à esquerda («), deslocamento à direita (»), ou exclusivo, multiplicação, divisão, resto e negação. Também a instrução mover (*mov*) foi adicionada como uma operação especial da ULA para ambas as classes nos dois modos de operandos. BPF_ALU64 também contém um deslocamento à direita com sinal. Além disso, BPF_ALU contém instruções de conversão de *endianness* para meia palavra, palavra, palavra dupla em um dado registro de fonte.

BPF_JMP: Esta classe é dedicada a operações de salto. Os saltos podem ser incondicionais e condicionais. Saltos incondicionais simplesmente movem o contador de programa para frente, de modo que a próxima instrução a ser executada em relação à instrução atual seja *deslocamento + 1*, onde o deslocamento é codificado na instrução. Como deslocamento tem sinal (*signed)*, o salto também pode ser executado para trás, desde que esteja dentro dos limites do programa. Os saltos condicionais operam em operandos de origem baseados em registro e baseados em imediato. Se a condição nas operações de salto resultar em verdadeiro, então um salto relativo para desvio + 1 é realizado, caso contrário, a próxima instrução (0 + 1) é executada. Essa lógica de salto é diferente em comparação com o BPF e permite uma melhor previsão de ramificação, pois ela se ajusta à lógica do preditor de desvios da CPU com mais naturalidade. As condições disponíveis são igualdade (jeq ==), diferença (jne !=), maior (jgt >), maior ou igual (jge >=), maior

com sinal (jsgt >), maior ou igual com sinal (jsge >=), menor (jlt <), menor ou igual (jle <=), menor com sinal (jslt <), menor ou igual com sinal (jsle <=) e jset (salto se origem e destino). Além disso, há três operações de salto especiais dentro dessa classe: a instrução de saída (*exit*) que deixará o programa BPF e retornará o valor atual em r0 como um código de retorno, a instrução de chamada *call*, que emitirá uma chamada de função em uma das funções auxiliares do eBPF e uma instrução de chamada cauda *tail*, que saltará para um outro programa eBPF.

3.2.2. Código de byte do eBPF

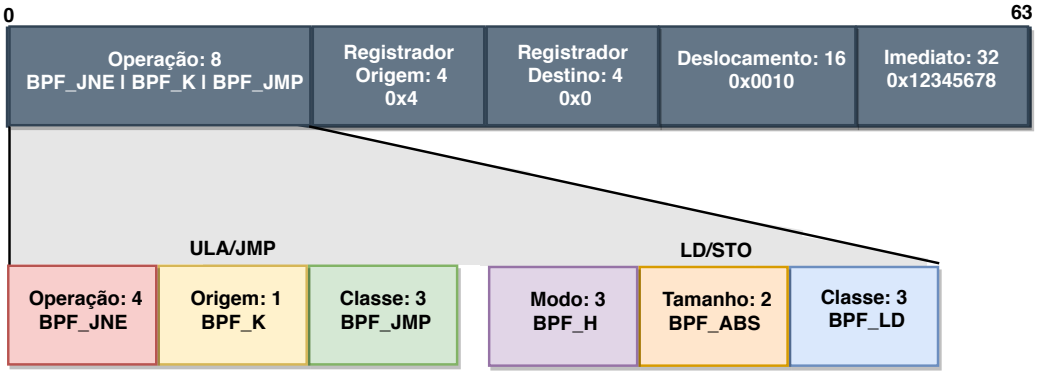


Figura 3.2. Código de byte eBPF.

A figura 3.2 mostra o código de byte do eBPF. O processador eBPF possui instruções de 64 bits, sendo 32 bits para representação de um número imediato (`imm`), 16 bits para o `offset`, 4 bits para registrador de origem, 4 bits para registrador de destino e, por fim, o código da operação (`opcode`) de 8 bits. O opcode pode ser redividido dependendo da classe da instrução. Para *load (LD)* e *store (STO)*, o opcode possui 3 bits mais significativos para modo de acesso à memória, 2 bits para tamanho da palavra e 3 bits para classe de instrução. Para as instruções de desvio *jump (JMP)*, e que utilizam a unidade lógica e aritmética (ULA), os 4 bits mais significativos especificam qual é a operação, 1 bit para especificar se a instrução é realizada com o valor imediato ou com o operando de origem e por fim 3 bits menos significativos para a classe da instrução.

3.2.3. Conjunto de Registradores

A tabela 3.1 descreve a funcionalidade de cada registradores eBPF. O registrador r10 é apenas de leitura e serve para armazenar o ponteiro do quadro para acesso à pilha. O registrador r0 é onde armazena o valor de retorno de função. É nesse registrador que fica armazeno ao final da computação qual a ação que será feita no encaminhamento do pacote.

3.3. Sistema eBPF

O sistema eBPF é composto por uma série de componentes responsáveis por compilar, verificar e executar códigos fonte de aplicações desenvolvidas. Mais detalhes sobre eles são dados a seguir.

Tabela 3.1. Descrição do conjunto de registradores do eBPF

Registrador	Descrição
R0	valor de retorno de funções e da saída do programa eBPF.
R1 - R5	passagem de argumento de funções.
R6 - R9	registradores que preservam valores em chamadas de função.
R10	ponteiro do quadro para acesso a pilha. É apenas de leitura.

3.3.1. Visão Geral

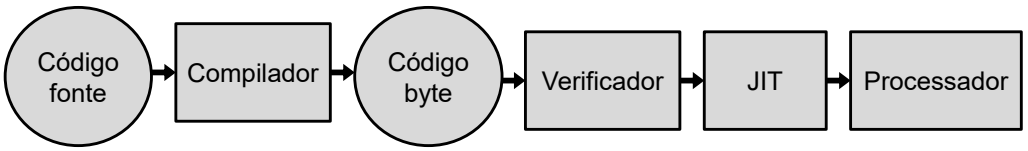


Figura 3.3. Fluxo de Trabalho para o eBPF.

O fluxo de trabalho típico do sistema eBPF é ilustrado na figura 3.3. Os programas eBPF são escritos em linguagem de alto nível (C, P4, Go), compilados pelo LLVM em arquivos objeto/ELF, que são analisados pelo carregador BPF ELF do espaço do usuário (como *iproute2* ou outros) e inseridos no *kernel* por meio da chamada do sistema BPF. O *kernel* verifica as instruções eBPF e faz a tradução dinâmica (JIT), retornando um novo descritor de arquivo para o programa. Este descritor pode então ser anexado a um subsistema (por exemplo, rede). Se suportado, o subsistema poderia, então, transferir o programa BPF para o hardware (por exemplo, a placa de rede) senão é executado pelo próprio processador.

3.3.2. Compilador

Linguagens de alto nível podem ser utilizadas para escrever código para o plano de dados que pode ser compilado para o conjunto de instruções do eBPF. O compilador LLVM 3.9 possui um *backend* para a plataforma eBPF, permitindo programar em um subconjunto de C e gerando código executável no formato eBPF.

Também é possível gerar instruções eBPF através de linguagens de domínio específico, como por exemplo, P4 [Bosshart et al. 2014]. Existem esforços do projeto de código aberto IOVisor [IOvisor 2019] e da VMWare [VMWare 2018] que já implementaram um compilador de P4 para eBPF [Budiu 2015].

3.3.2.1. Subconjunto de C

É possível programar para eBPF usando um subconjunto da linguagem C. Esse subconjunto exclui algumas bibliotecas externas, chamadas de sistemas e aritmética de ponteiro enquanto provê funções para definição e manipulação de tabelas.

Existem alguns detalhes importantes:

- O eBPF só pode utilizar um subconjunto de bibliotecas da linguagem C. Por exem-

plo, a função *printf()* não está disponível para a utilização. Neste caso, é necessário definir e utilizar funções auxiliares.

- Ainda não se tem noção de chamada de funções. Logo, todas as chamadas de funções são *inline*. Neste caso, basta declara a função como *inline*.
- Não se pode ter nenhuma variável global. A solução é utilizar mapas.
- Devido ao verificador, (ainda) não se pode ter laços (*loops*) a menos que sejam desenrolados através da diretiva de compilação *#pragma clang loop unroll (full)*. Outra solução é desenvolver um sistema eBPF mas que não inclua o verificador.
- Ainda não se tem suporte a cadeia de caracteres (*strings*).
- O espaço de pilha é limitado até 512 bytes.

3.3.3. Verificador

Para garantir a integridade e segurança do sistema operacional, o *kernel* do Linux utiliza um verificador que faz a análise estática de programas com instruções eBPF sendo carregados no sistema. A sua implementação é feita no arquivo `kernel/bpf/verifier.c`.

O verificador permite checar a validade, segurança e desempenho dos programas eBPF. Entre outras coisas, ele verifica se um programa termina, se os acessos de memória estão nos intervalo de memória permitidos para o programa e qual a maior profundidade do caminho de execução. O verificador é chamado depois do código ter sido compilado e antes de carregá-lo no plano de dados. Uma boa descrição geral é apresentada em [Høiland-Jørgensen et al. 2018].

A verificação da condição de término do programa é feita pelo verificador através da geração de um grafo acíclico direcionado (GAD). Programas eBPF que não possuem saltos para trás ou que possuem apenas laços de tamanho pré-definidos (podendo assim ter o laço desenrolado (*loop unroll*)) podem ser sintetizado em um GAD e isso garante o término deles. Um exemplo de geração de GAD é dado a seguir.

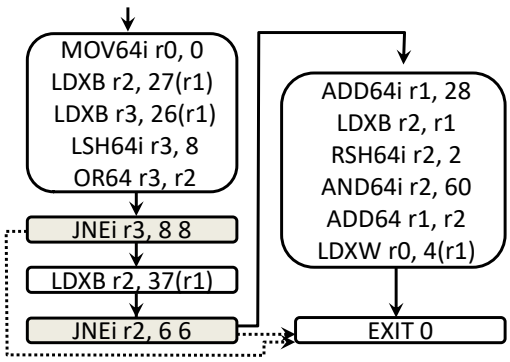


Figura 3.4. Exemplo de Grafo Acíclico Direcionado para o código eBPF.

O código 2 ilustra um exemplo de código na linguagem C para acessar o campo número de sequência do protocolo TCP. A figura 3.4 mostra o respectivo GAD para o

Código 2 Exemplo de código em C para acessar o número de sequência TCP.

```
#include <linux/if_ether.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include "ebpf_switch.h"

uint64_t prog(struct packet *pkt){
  if (pkt->metadata.in_port == 0) {

    // Verifica se o quadro Ethernet possui um pacote ipv4
    if (pkt->eth.h_proto == 0x0008) {
      struct ip *ipv4 = (struct ip *)
        ( ((uint8_t *)&pkt->eth) + ETH_HLEN );

      // Verifa se pacote IPv4 possui segmento TCP
      if (ipv4->ip_p == 6) {
        struct tcphdr *tcp = (struct tcphdr *)
          ( ((uint32_t *) ipv4)+ ipv4->ip_hl );
     // retorna número de sequência
        return tcp->th_seq;
      }
    }
  }
  return 0;
}
```

código presente no código 2. Cada nó do GAD contém uma ou mais instruções do eBPF. Instruções terminando com a letra i indicam o uso de valores imediatos.

Os nós de saltos condicionais são aqules que contêm duas linhas de saída e plano de fundo cinza claro. Linha sólida indica o próximo nó do ACFG. Linha pontilhada indica um salto para outro nó do ACFG. No exemplo dado, os nós de salto condicionais contêm a instrução desvio não igual (*jump not equal (jnei)*). O último valor da instrução jnei indica o número instruções para pular quando a condição é válida.

Para entender melhor o programa eBPF, vale lembrar que o registrador R1 começa com um ponteiro para o pacote armazenado na memória de dados e o registrador R0 armazena o valor de retorno, conforme descrito na tabela 3.1.

No primeiro nó, as instruções de byte de carga extraem os bytes 26 e 27 da memória de dados. Dado que os metadados têm 16 bytes, isso representa os bytes 10 e 11 do pacote (contando a partir de 0), que é o campo do tipo Ethernet. Em seguida, uma troca de bytes é feita para definir a ordem dos bytes (*endianess*). A instrução de primeiro salto compara se o tipo de Ethernet é 0x0800. Em seguida, o campo do protocolo IP é extraído do pacote IPv4 e verifica-se se é o protocolo TCP (valor 6). Finalmente, o número de sequência do TCP é extraído e colocado no registrador de retorno (R0). A última instrução informa que o código termina. O GAD é útil para determinar o pior tempo de execução do caso.

3.3.4. Pseudo-sistema de arquivos

O sistema BPF apresenta um pseudo-sistema de arquivos localizado em `/sys/fs/bpf`. Através dele é possível ter acesso aos mapas (seção 3.5.2) de programas BPF carregados no sistema.

3.4. Ganchos (Hooks)

Gancho (*Hook*) é uma interface que permite que um programador insira programação customizada no *kernel*. O gancho permite modificar ou melhorar o comportamento de um sistema operacional, aplicações ou outros componentes de software através da interceptação de chamadas de funções, mensagens ou eventos passados entre componentes de software.

Em Redes de Computadores, ganchos são utilizados para a interceptação de pacotes antes da chamada ou durante a execução no sistema operacional. O processamento da pilha de rede do sistema operacional pode ser custoso. O gancho permite que aplicações que executam no espaço de usuário possam processar pacotes evitando passar nessa pilha e, assim, diminuir o custo adicional de processar pacotes.

3.4.1. Tipos de ganchos

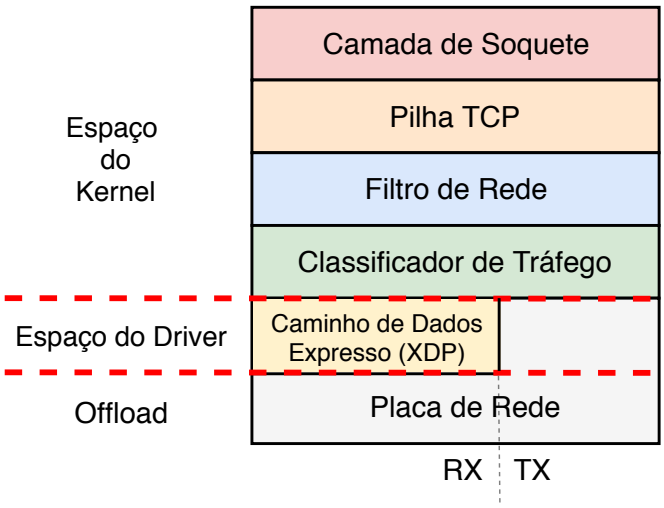


Figura 3.5. Camadas para ganchos.

Os pacotes de entrada são transmitidos através das camadas do *kernel* da placa de rede, conforme mostrado na figura 3.5. As camadas são: camada de soquete, pilha TCP, filtro de rede (*Netfilter*), classificador de tráfego (*Traffic Control/Classifier (TC)*), caminho de dados expresso (*eXpress Data Path (XDP)*) e a placa de rede.

Os pacotes podem ser destinados a um aplicativo do espaço do usuário ou podem ser interceptados por um módulo do *kernel*, como `iptables`, localizados na camada de filtro da rede. Os programas eBPF podem se conectar a vários locais dentro do *kernel* do Linux e filtrar pacotes. O melhor desempenho pode ser obtido filtrando os pacotes nas camadas inferiores do *kernel*, resultando no gancho XDP ser a melhor opção em termos de desempenho.

3.4.2. O que é XDP

O gancho Caminho de Dados Expresso (*eXpress Data Path (XDP)*) ocorre dentro do driver do dispositivo de rede, permitindo que os pacotes sejam processados pela CPU no primeiro ponto. O gancho XDP também permite transferir a computação para a placa de rede.

3.4.3. Ações XDP

A tabela 3.2 descreve as ações do XDP, incluindo os valores, nome e descrição. Os pacotes serão encaminhados de acordo com o valor da ação. O valor de retorno do programa eBPF é armazenado no registrador 0. Esse valor de retorno determina qual é a ação XDP correspondente.

Tabela 3.2. Descrição do conjunto de ações do XDP

Valor	Ação	Descrição
0	XDP_ABORTED	Erro. Descarta o pacote.
1	XDP_DROP	Descarta o pacote.
2	XDP_PASS	Permite o pacote continuar até o *kernel*.
3	XDP_TX	Devolve o pacote para a rede.
4	XDP_REDIRECT	Redireciona o pacote.

As quatro primeiras ações são de retorno simples (sem parâmetros), que podem abortar (descarta o pacote e passa o *tracepoint trace_xdp_exception*), descartar pacote, permitir que ele seja processado pela pilha de rede do *kernel*, ou imediatamente retransmiti-lo pela mesma interface de rede. O XDP_REDIRECT permite que o programa XDP redirecione o pacote, oferecendo controle adicional sobre seu processamento posterior.

A ação de redirecionamento requer um parâmetro adicional que especifica o alvo de redirecionamento, que é definido através de uma função auxiliar antes do programa sair. A funcionalidade de redirecionamento pode ser usada (i) para transmitir o pacote em uma interface de rede diferente (incluindo interfaces virtuais conectadas para máquinas virtuais), (ii) para passá-lo para uma CPU diferente para processamento adicional, ou (iii) para passá-lo diretamente para um soquete (AF_XDP) no espaço de usuário. Além disso, como o parâmetro de redirecionamento é implementado como uma pesquisa de mapa (onde o programa XDP fornece a chave de pesquisa), as regras de redirecionamento podem ser alteradas dinamicamente sem modificar programa.

3.4.4. Modos de Operação do XDP

Visando obter alto desempenho, programas eBPF carregados ao gancho XDP alteram o processamento de pacotes em nível de placa de rede, o que requer suporte explícito do driver associado. Alguns drivers para dispositivos de alta velocidade como `i40e`, `nfp`, `mlx*` e a família `ixgbe` já suportam essa funcionalidade, também chamada de *XDP nativo*. Nesses dispositivos programas eBPF são executados diretamente pelo driver, antes mesmo do pacote ser entregue ao sistema operacional. Uma lista atualizada dos drivers com suporte ao XDP nativo é mantida pelo projeto BCC [BCC 2019d].

No entanto, o *kernel* do Linux também oferece suporte para dispositivos que não

provêm essa funcionalidade em nível de driver através do modo *XDP genérico*. Nesse modo são criados buffers de soquete(*socket buffers*) especiais assim que os pacotes são recebidos pelo sistema operacional e a execução de programas XDP é feita pelo próprio *kernel*, emulando a execução do XDP nativo. Dessa forma até mesmo dispositivos que não têm suporte nativo ao XDP podem executar programas eBPF nesse gancho, ao custo de desempenho inferior por conta da alocação dos buffers [Miano et al. 2018].

A escolha entre esses dois modos de operação é feita automaticamente pelo sistema durante o carregamento do programa eBPF. Uma vez carregado, é possível verificar o modo de operação utilizando a ferramenta *iproute2* como mostrado na seção a seguir.

Ainda existe um terceiro modo de operação, chamado *XDP offload*. Como o nome sugere, o programa eBPF é descarregado para a placa de rede para ser executado em hardware, alcançando desempenho superior aos outros dois modos. Para habilitar esse modo, é necessário indicar explicitamente durante o carregamento do programa, além de um dispositivo capaz de executar eBPF em hardware. Atualmente as placas de rede inteligentes da empresa Netronome são os únicos dispositivos no mercado com essa capacidade.

3.4.5. Exemplo de gancho XDP e XDP offload

O exemplo a seguir requer um *kernel* atualizado a partir do Ubuntu 18.04 ou Fedora 28.

Para começar, escreve-se um programa C simples para processar pacotes de entrada e retornar XDP_DROP, o que impedirá que todos os pacotes recebidos cheguem ao host.

```c
#include <linux/bpf.h>
int main(){
    return XDP_DROP;
}
```

Depois de salvar o arquivo como xdp.c, o código pode ser compilado em um objeto eBPF usando o comando clang descrito a seguir. O compilador clang contém um *backend* eBPF permitindo que o código de montagem do eBPF seja gerado.

```
$ clang -target bpf -O2 -c xdp.c -o xdp.o
```

Depois que o programa é compilado, ele pode ser carregado usando a ferramenta *ip link* (altere [DEV] para o nome da interface relevante). No exemplo, o código não contém um rótulo de seção, portanto, o nome da seção ELF padrão .text é usado para o carregamento.

```
$ ip -force link set dev [DEV] xdpdrv obj xdp.o sec .text
```

Depois que o programa for carregado, o ip link mostrará que o programa eBPF está conectado à interface no gancho do XDP.

```
$ ip link show dev [DEV]
6: DEV: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 xdp qdisc
mq state UP mode DEFAULT group default qlen 1000
    link/ether 00:15:4d:13:08:80 brd ff:ff:ff:ff:ff:ff
    prog/xdp id 27 tag f95672269956c10d jited
```

A palavra `xdp` na primeira linha da saída do comando indica que o programa está carregado no modo *XDP nativo*. Outros valores possíveis são `xdpgeneric` e `xdpoffload` para os modos *XDP genérico* e *XDP offload*, respectivamente. Para descarregar o programa XDP, um comando ip link semelhante pode ser usado, mas com o parâmetro off.

```
$ ip link set dev [DEV] xdpdrv off
```

3.4.5.1. Transferindo a computação para a placa de rede

No exemplo anterior, o eBPF foi executado no gancho XDP que está localizado dentro do driver e utiliza a CPU host x86 para o manuseio de pacotes. Se a interface de rede suportar, pode-se transferir a computação (descarregamento) do programa eBPF diretamente no processador da placa de rede. Atualmente, o único hardware com esse suporte é o Netronome Agilio CX SmartNIC.

Para descarregar o programa, o mesmo método pode ser utilizado, a única exceção é que *xdpoffload* é usado dentro do comando *ip link*.

```
$ ip -force link set dev [DEV] xdpoffload obj xdp.o sec .text
```

Para descarregar o programa, defina *xdpoffload* como desligado (off).

```
$ ip link set dev [DEV] xdpoffload off
```

3.4.6. Comparação entre XDP e DPDK

O XDP às vezes é comparado ao DPDK. O XDP oferece outra opção para usuários que desejam desempenho e ainda aproveitam a capacidade de programação do *kernel*. Algumas das funções que o XDP oferece incluem o seguinte:

- Remove a necessidade de código e licenciamento de terceiros
- Permite a opção de varredura ou interrupção
- Remove a necessidade de alocar páginas grandes
- Remove a necessidade de CPUs dedicadas, pois os usuários têm mais opções na estruturação de trabalho entre CPUs

- Remove a necessidade de injetar pacotes no *kernel* de um aplicativo de espaço do usuário de terceiros
- Remove a necessidade de definir um novo modelo de segurança para acessar o hardware de rede

3.4.7. Gancho *Traffic Control* (TC)

A pilha de rede do *kernel* do Linux possui uma camada para executar políticas de controle de tráfego, a camada de *Traffic Control* (TC). Nela o administrador de rede é capaz de configurar diferentes disciplinas para as diversas filas de pacotes do sistema operacional, assim como adicionar filtros para recusar ou modificar pacotes.

O TC apresenta um tipo de disciplina especial chamada `clsact`. Ela permite que as ações de processamento da fila sejam definidas por um programa eBPF. O pacote a ser processado é entregue para o programa eBPF configurado, no qual pode ser modificado, e o valor de retorno indica para o TC qual ação deve ser tomada pela fila. Os valores de retorno disponíveis estão definidos no arquivo `linux/pkt_cls.h`, sendo os mais comuns listados na tabela 3.3.

Tabela 3.3. Descrição do conjunto de ações do TC

Valor	Ação	Descrição
0	TC_ACT_OK	Dá prosseguimento ao pacote na fila do TC.
2	TC_ACT_SHOT	Descarta o pacote.
-1	TC_ACT_UNSPEC	Usa a ação padrão do TC.
3	TC_ACT_PIPE	Executa a próxima ação, se existir.
1	TC_ACT_RECLASSIFY	Reinicia a classificação desde o início.

O carregamento de programas no gancho TC é feito utilizando a ferramenta `tc`, disponível no pacote `iproute2`. O comando a seguir ilustra como criar a discplina `clsact` e carregar um código eBPF para processar os pacotes na interface `eth0`:

```
$ tc qdisc add dev eth0 clsact
$ tc filter add dev eth0 <direction> bpf da obj <ebpf-obj>
  ↪ sec <section>
```

O parâmetro `<direction>` indica a qual direção o programa deve ser associado, podendo ser `ingress` para RX ou `egress` para TX. `<ebpf-obj>` e `<section>` devem ser os nomes do arquivo que contém o código eBPF compilado e da seção correspondente ao programa que deseja ser adicionado, respectivamente.

Para verificar se há algum programa já carregado na interface `eth0`, basta utilizar o seguinte comando:

```
$ tc filter show dev eth0 <direction>
```

3.5. Programas eBPF

O eBPF suporta diferentes tipos de aplicações, por exemplo, medições de desempenho, filtragem e classificação de tráfego. Uma das principais vantagens do eBPF é fornecer um ambiente de programação flexível e seguro, onde programas podem ser escritos para diferentes contextos no *kernel* (*kernel probes*, eventos *perf*, dentre outros) [Maguire 2019].

Todo programa eBPF é verificado no momento em que é carregado para garantir que ele possa ser executado sem riscos dentro do *kernel*. Além disso, o eBPF suporta a compilação JIT de seu código de bytes para o conjunto de instruções nativo da máquina, o que torna os programas rápidos.

Programas eBPF possuem tipos que determinam os contextos em que eles podem executar. Eles utilizam estruturas de dados chamadas de mapas para armazenarem tipos de dados diferentes. Manipulação dos dados armazenados e realização de interações com o *kernel* são feitas através de funções especiais, chamadas de funções auxiliares.

3.5.1. Tipos de programas

O tipo de um programa eBPF define três pontos importantes: qual é a entrada do programa (o contexto), quais funções auxiliares ele pode utilizar e onde ele será carregado no *kernel*. Por exemplo, a entrada para um programa de filtragem de soquete é um pacote da rede, enquanto para um programa de rastreamento é um conjunto de valores dos registradores. De forma similar, o subconjunto das funções auxiliares que um programa de filtragem de soquete pode utilizar não é igual a um programa de rastreamento, embora possam existir funções em comum entre eles.

Os tipos de programas eBPF suportados são definidos através da *enum* `bpf_prog_type` em bpf.h, e podem crescer com novas versões do *kernel*. Na versão 5.0 do *kernel*, são disponibilizados um total de 23 tipos de programas diferentes. Alguns deles são:

- `BPF_PROG_TYPE_SOCKET_FILTER`: programa de filtragem de soquete;
- `BPF_PROG_TYPE_KPROBE`: programa para instrumentação em funções do *kernel* via *kprobe*;
- `BPF_PROG_TYPE_SCHED_CLS`: programa de classificação de controle de tráfego de rede;
- `BPF_PROG_TYPE_XDP`: programa XDP;
- `BPF_PROG_TYPE_PERF_EVENT`: programa para instrumentação de eventos *perf* de software e hardware;
- `BPF_PROG_TYPE_LWT_IN`, `BPF_PROG_TYPE_LWT_OUT` e `BPF_PROG_TYPE_LWT_XMIT`: programas para implementação de túneis leves (*lightweight tunnels*).
- `BPF_PROG_TYPE_SOCK_OPS`: programa que permite a obtenção de operações de soquetes (estabelecimento de conexão, tempo limite de retransmissão etc) e a definição de parâmetros de soquetes;

- `BPF_PROG_TYPE_SK_SKB`: programa que permite o acesso ao *skb* (*socket buffer*) e a detalhes de soquetes (endereço IP, porta etc) com o objetivo de realizar redirecionamento entre soquetes.

A lista dos tipos de programas suportados pode ser obtida do código fonte do Linux utilizando o seguinte comando:

```
$ git grep -W 'bpf_prog_type {' include/uapi/linux/bpf.h
```

3.5.2. Mapas

Mapas são estruturas de dados genéricas com a função de armazenar diferentes tipos de dados através de pares chave e valor. Tipos de dados são tratados como *blobs* binários e cabe ao usuário definir o tamanho das chaves e valores durante a criação do mapa.

Mapas eBPF podem ser criados de duas formas: por programas de espaço de usuário utilizando a chamada de sistema `bpf`, podendo ser manipulados através do descritor de arquivo retornado por ela em caso de sucesso, ou diretamente por programas eBPF carregados no *kernel*. No primeiro caso, a criação é feita com a utilização do comando `BPF_MAP_CREATE` (do *enum* `bpf_cmd`) e de uma *union* de configuração `bpf_attr` como parâmetros da chamada de sistema

```
bpf(BPF_MAP_CREATE, &bpf_attr, sizeof(bpf_attr)).
```

A *union* de configuração requer a definição dos seguintes atributos:

1. *map_type*: tipo do mapa a ser criado;
2. *key_size*: tamanho das chaves em bytes;
3. *value_size*: tamanho dos valores em bytes;
4. *max_entries*: quantidade máxima de entradas no mapa.

Um processo no espaço de usuário pode criar múltiplos mapas. Mapas podem ser acessados paralelamente por outros programas eBPF. Isso permite o compartilhamento de dados entre aplicações eBPF dentro *kernel*, e entre aplicações no *kernel* e espaço de usuário. Para destruir um mapa, o descritor de arquivo associado a ele deve ser fechado.

Cada programa eBPF deve declarar os mapas utilizados como variáveis globais na seção *maps* utilizando a estrutura `bpf_map_def` definida pela `libbpf` (§3.6.1) . O exemplo a seguir declara um mapa `rxcnt` do tipo `BPF_PROG_TYPE_PERCPU_ARRAY`:

```
struct bpf_map_def SEC("maps") rxcnt = {
    .type = BPF_MAP_TYPE_PERCPU_ARRAY,
    .key_size = sizeof(uint32_t),
    .value_size = sizeof(long),
    .max_entries = 256,
};
```

3.5.3. Tipos de mapas

Existem diferentes tipos de mapas disponíveis para programas eBPF definidos no *enum* `bpf_map_type`. Cada tipo oferece um determinado comportamento, alguns deles são utilizados de forma genérica, outros para casos de uso específicos.

Exemplos de código de mapas eBPF são providas pelo *kernel* do Linux. A versão 5.0 do *kernel* lista um total de 23 mapas diferentes. Alguns deles são descritos:

- `BPF_MAP_TYPE_HASH`: tabela hash genérica.
- `BPF_MAP_TYPE_ARRAY`: vetor otimizado para pesquisas rápidas. É frequentemente utilizado por aplicações que utilizam contadores.
- `BPF_MAP_TYPE_PROG_ARRAY`: vetor de descritores de arquivos correspondentes a programas eBPF. Sua utilização permite, por exemplo, a chamada de subprogramas para lidar com situações específicas.
- `BPF_MAP_TYPE_PERCPU_HASH`: mapa similar ao `BPF_MAP_TYPE_HASH`. Permite a criação de uma tabela hash para cada núcleo do processador.
- `BPF_MAP_TYPE_PERCPU_ARRAY`: mapa similar ao `BPF_MAP_TYPE_ARRAY`. Permite a criação de um vetor para cada núcleo do processador.
- `BPF_MAP_TYPE_PERF_EVENT_ARRAY`: mapa para o armazenamento e leitura de contadores de eventos da ferramenta *perf*.
- `BPF_MAP_TYPE_LRU_HASH`: tabela hash que mantém somente os elementos utilizados pela última vez recentemente (LRU). Assim, quando a tabela se encontra cheia, os elementos utilizados pela última vez a mais tempo são removidos primeiramente.
- `BPF_MAP_TYPE_LRU_PERCPU_HASH`: versão que utiliza tabelas hash LRU do mapa `BPF_MAP_TYPE_PERCPU_HASH`.
- `BPF_MAP_TYPE_LPM_TRIE`: uma *trie* para o casamento do prefixo mais longo.
- `BPF_MAP_TYPE_STACK_TRACE`: mapa para armazenamento de *traces* de pilhas.
- `BPF_MAP_TYPE_ARRAY_OF_MAPS`: vetor para o armazenamento de mapas eBPF.
- `BPF_MAP_TYPE_HASH_OF_MAPS`: tabela hash para armazenamento de mapas eBPF.
- `BPF_MAP_TYPE_DEVMAP`: mapa para armazenamento e leitura de referências a dispositivos de rede.
- `BPF_MAP_TYPE_SOCKMAP`: mapa para armazenamento de soquetes. Pode ser utilizado para implementar redirecionamento de soquetes, por exemplo.
- `BPF_MAP_TYPE_QUEUE`: mapa com comportamento similar ao de uma fila.

- `BPF_MAP_TYPE_STACK`: mapa com comportamento similar ao de uma pilha.

A lista de todos os tipos de mapas suportados pode ser obtida diretamente do código fonte do Linux a partir do seguinte comando:

```
$ git grep -W 'bpf_map_type {' include/uapi/linux/bpf.h
```

3.5.4. Funções auxiliares

O eBPF se diferencia do BPF "clássico"(cBPF) em muitos aspectos. Um deles é a habilidade de permitir que programas façam a chamada de funções especiais denominadas funções auxiliares. Funções auxiliares permitem que programas eBPF sejam capazes de interagir com o sistema e com o contexto ao qual essas funções trabalham. Exemplos de tarefas realizadas por funções auxiliares incluem interação com mapas eBPF, manipulação de pacotes da rede e impressão de mensagens de depuração.

As funções auxiliares disponíveis para os programas eBPF são restritas as funções definidas no *kernel*. Como existem diferentes tipos de programa eBPF e eles não executam em um mesmo contexto, cada tipo de programa pode chamar somente um subconjunto dessas funções. Por exemplo, algumas funções auxiliares só estão disponíveis para programas XDP. O projeto BCC oferece uma documentação atualizada das funções auxiliares disponíveis para cada tipo de programa eBPF [BCC 2019b].

Novas funções auxiliares podem ser adicionadas somente através de extensões do *kernel*. Isso significa que nenhuma função auxiliar pode ser estendida ou adicionada através da adição de módulos no *kernel*. Inserir uma nova função auxiliar requer o seguimento de uma assinatura convencionada pelo eBPF. Essa assinatura é compartilhada por todas as funções auxiliares, limitando a quantidade de argumentos para no máximo cinco. Uma assinatura é definida como:

```
u64 fn(u64 r1, u64 r2, u64 r3, u64 r4, u64 r5)
```

Toda nova função auxiliar adicionada é compilada em tempo de execução de forma transparente e eficiente. Isso significa que o compilador JIT necessita somente emitir uma instrução de chamada, pois o mapeamento dos registradores eBPF é feito de forma a corresponder com a convenção de chamada da arquitetura subjacente. Programas eBPF chamam diretamente as funções auxiliares compiladas e por isso elas não introduzem nenhum *overhead*, oferecendo um bom desempenho.

O número de funções auxiliares atualmente disponíveis é amplo e cresce ao longo do tempo. A versão 5.0 do *kernel* disponibiliza um total de 92 funções diferentes. Algumas delas são descritas:

- `bpf_map_lookup_elem`, `bpf_map_update_elem` e `bpf_map_delete_elem`: funções utilizadas, respectivamente, para pesquisar, atualizar e remover uma entrada associada a uma chave no mapa.
- `bpf_get_prandom_u32`: retorna um valor de 32 bits pseudo-aleatório.

- `bpf_tail_call`: função utilizada para pular para outro programa eBPF.
- `bpf_l3_csum_replace` e `bpf_l4_csum_replace`: funções utilizadas para recalcular o *checksum* das camadas 3 (ex. IP) e 4 (ex. TCP ou UDP) de pacotes.
- `bpf_ktime_get_ns`: retorna o tempo decorrido desde o *boot* do sistema, em nanosegundos.
- `bpf_clone_redirect` e `bpf_redirect`: funções utilizadas para redirecionar pacotes para um dispositivo de rede. A diferença entre elas é que a primeira clona o pacote, enquanto a segunda não (o que garante melhor desempenho).
- `bpf_skb_vlan_push` e `bpf_skb_vlan_pop`: funções para a adição (*push*) e remoção (*pop*) de um cabeçalho VLAN de um pacote.
- `bpf_getsockopt` e `bpf_setsockopt`: funções que emulam as respectivas chamadas de `getsockopt()` e `setsockopt()` para soquetes.
- `get_local_storage`: retorna um ponteiro para uma área de armazenamento local. Dependendo do tipo do programa eBPF, essa área de armazenamento pode ser compartilhada entre múltiplas instâncias do programa rodando paralelamente.

A declaração de todas as funções auxiliares é feita no arquivo `bpf_helpers.h`, localizado no diretório `tools/testing/selftests/bpf`. Outras operações comuns durante o processamento de pacotes como a conversão de valores binários entre *little* e *big endian* estão disponíveis no arquivo `bpf_endian.h`, localizado no mesmo diretório. Este arquivo oferece versões de funções conhecidas como `ntohs` e `htons`, na forma de funções auxiliares (`bpf_ntohs` e `bpf_htons`, respectivamente), assim como outras funções similares.

Além de incluir essas bibliotecas no código fonte do programa eBPF, é necessário passar o caminho do diretório onde elas estão localizadas para que o compilador saiba como importar os arquivos. Isso pode ser feito utilizando a *flag* `-I` do *clang*. Caso não deseje trabalhar diretamente com o código do *kernel* (ex: evitar dependências externas), é possível usar cópias locais destes dois arquivos. A geração do código de máquina das funções auxiliares é feita durante a compilação do próprio *kernel* e as chamadas em tempo de execução são feitas internamente pelo sistema operacional. Por conta disso, para gerar o executável de um programa eBPF o compilador só necessita saber a assinatura das funções auxiliares, possibilitando o uso deste artifício.

3.5.5. Exemplos

O código fonte do Linux contém diversos exemplos de programas eBPF, localizados em dois diretórios distintos: `samples/bpf` e `tools/testing/selftests/bpf`. Ambos contém programas que demonstram o uso de diversas funcionalidades e ganchos da pilha do *kernel*, com novos programas sendo adicionados a cada nova versão do *kernel*.

A maior parte dos exemplos do primeiro diretório está dividida em dois arquivos separados, um com código em espaço de usuário para carregar o programa (arquivos

terminados em *user.c*) e o outro com a implementação do programa eBPF a ser carregado no *kernel* (arquivos terminados em *kern.c*).

Já os exemplos do segundo diretório são utilizados como base para a execução de testes funcionais durante o desenvolvimento do arcabouço eBPF. Por conta disso, a tendência da comunidade de desenvolvedores é adicionar os programas mais recentes ao segundo diretório, no formato de testes de unidade. Em geral estes exemplos utilizam as últimas novidades, representando melhor o estado atual e as ferramentas disponíveis para o desenvolvimento de programas eBPF.

A seguir é apresentado o exemplo `xdp1`, presente no diretório `samples/bpf`. O arquivo `xdp1_kern.c` contém um programa eBPF que processa pacotes no gancho XDP e armazena a contagem do número de pacotes recebidos por protocolo de camada de rede em um mapa.

```
8  #include <uapi/linux/bpf.h>
9  #include <linux/in.h>
10 #include <linux/if_ether.h>
11 #include <linux/if_packet.h>
12 #include <linux/if_vlan.h>
13 #include <linux/ip.h>
14 #include <linux/ipv6.h>
15 #include "bpf_helpers.h"
16
17 struct bpf_map_def SEC("maps") rxcnt = {
18     .type = BPF_MAP_TYPE_PERCPU_ARRAY,
19     .key_size = sizeof(u32),
20     .value_size = sizeof(long),
21     .max_entries = 256,
22 };
23
24 static int parse_ipv4(void *data, u64 nh_off, void *data_end) {
25     struct iphdr *iph = data + nh_off;
26
27     if (iph + 1 > data_end)
28         return 0;
29     return iph->protocol;
30 }
31
32 static int parse_ipv6(void *data, u64 nh_off, void *data_end) {
33     struct ipv6hdr *ip6h = data + nh_off;
34
35     if (ip6h + 1 > data_end)
36         return 0;
37     return ip6h->nexthdr;
38 }
39
40 SEC("xdp1")
41 int xdp_prog1(struct xdp_md *ctx) {
42     void *data_end = (void *)(long)ctx->data_end;
43     void *data = (void *)(long)ctx->data;
44     struct ethhdr *eth = data;
45     int rc = XDP_DROP;
46     long *value;
```

```
47      u16 h_proto;
48      u64 nh_off;
49      u32 ipproto;
50
51      nh_off = sizeof(*eth);
52      if (data + nh_off > data_end)
53          return rc;
54
55      h_proto = eth->h_proto;
56
57      if (h_proto == htons(ETH_P_8021Q)  h_proto == htons(ETH_P_8021AD))
          ↪  {
58          struct vlan_hdr *vhdr;
59
60          vhdr = data + nh_off;
61          nh_off += sizeof(struct vlan_hdr);
62          if (data + nh_off > data_end)
63              return rc;
64          h_proto = vhdr->h_vlan_encapsulated_proto;
65      }
66      if (h_proto == htons(ETH_P_8021Q)  h_proto == htons(ETH_P_8021AD))
          ↪  {
67          struct vlan_hdr *vhdr;
68
69          vhdr = data + nh_off;
70          nh_off += sizeof(struct vlan_hdr);
71          if (data + nh_off > data_end)
72              return rc;
73          h_proto = vhdr->h_vlan_encapsulated_proto;
74      }
75
76      if (h_proto == htons(ETH_P_IP))
77          ipproto = parse_ipv4(data, nh_off, data_end);
78      else if (h_proto == htons(ETH_P_IPV6))
79          ipproto = parse_ipv6(data, nh_off, data_end);
80      else
81          ipproto = 0;
82
83      value = bpf_map_lookup_elem(&rxcnt, &ipproto);
84      if (value)
85          *value += 1;
86
87      return rc;
88  }
89  char _license[] SEC("license") = "GPL";
```

Múltiplos programas eBPF podem residir no mesmo arquivo `.c`, sendo separados em diferentes seções no arquivo ELF gerado pelo compilador. O rótulo de seção acima da função correspondente (Linha 40) indica para o compilador o nome da seção ELF que conterá o programa no arquivo objeto gerado. Essa informação é necessária durante o carregamento do código no *kernel* para que o sistema saiba qual seção ELF deve ser carregada. Caso o arquivo contenha apenas um programa, o rótulo pode ser omitido, e o programa será carregado na seção padrão `.text`. Rótulos de seção também são utilizados durante a declaração de mapas (Linha 17) e da licença do programa (Linha 89),

sendo ambos obrigatórios e com valores fixos. A seção de licença é usada pelo verificador para determinar quais funções auxiliares estarão disponíveis para o usuário, pois algumas funções só podem ser usadas por programas que declaram a mesma licença.

O programa em questão foi desenvolvido para ser carregado no gancho XDP, portanto o parâmetro de entrada da função deve ser do tipo `struct xdp_md`, representando o contexto passado por esse gancho para o programa eBPF. Os bytes do pacote sendo processado são delimitados pelos ponteiros `data` e `data_end`, que devem ser usados durante todo o programa para efetuar acessos ao pacote. As conversões de tipos para esses dois valores são padrão, de forma que as Linhas 42 e 43 devem ser utilizadas no início de todo programa eBPF.

Para garantir a integridade e a segurança do *kernel*, o verificador do sistema eBPF (§3.3.3) não permite acessos de memória além das variáveis locais e dos limites do pacote indicados pelo contexto passado. Para acessar quaisquer bytes do pacote, sempre é necessário fazer uma verificação de limites, como demonstrado nas Linhas 27, 35, 52, 62 e 71. Porém, cada byte só precisa ser verificado uma única vez, a não ser que funções auxiliares que modificam o espaço de armazenamento do pacote sejam usadas, como `bpf_xdp_adjust_head`. Nesse caso toda a checagem deve ser refeita após a chamada dessa função. Durante a análise do programa o verificador garante que todos os acessos de memória feitos ao pacote são em endereços verificados dessa forma. Caso esse tipo de checagem não seja feita dentro do programa eBPF, ele é rejeitado pelo verificador durante o carregamento no *kernel*.

Após determinar o tipo do protocolo de camada 3 do pacote, o programa atualiza o contador para o protocolo correspondente utilizando a função auxiliar de consulta (Linha 83). Essa função retorna um ponteiro para o valor atual armazenado no mapa, caso ele exista, ou `NULL` caso contrário. Esse endereço pode ser usado para alterar o dado armazenado de forma direta, sem a necessidade de uma operação de atualização do mapa. Por fim, o programa retorna a ação que deve ser tomada pelo gancho XDP para o pacote atual, que nesse caso é sempre `XDP_DROP`, indicando que o pacote deve ser descartado.

As estatísticas armazenadas no mapa `rxcnt` são então consultadas por uma aplicação em espaço de usuário, implementada pelo arquivo `xdp1_user.c`. Por questões de espaço destacamos abaixo apenas alguns trechos deste programa.

O programa inclui as bibliotecas `libbpf.h` (§3.6.1) e `bpf.h` para permitir interação com o sistema eBPF:

```
21 #include "bpf/bpf.h"
22 #include "bpf/libbpf.h"
```

A informações do programa a ser carregado são passadas por meio da estrutura `bpf_prog_load_attr`, incluindo o tipo de programa, o arquivo objeto contendo o programa e o identificador da interface ao qual ele deve ser associado.

```
73 struct bpf_prog_load_attr prog_load_attr = {
74     .prog_type        = BPF_PROG_TYPE_XDP,
75 };
```

```
112 snprintf(filename, sizeof(filename), "%s_kern.o", argv[0]);
113 prog_load_attr.file = filename;
```

Essa estrutura então é utilizada para carregar o programa no gancho XDP. Em caso de sucesso, após a chamada as variáveis `obj` e `prog_fd` contém as informações detalhadas do código já carregado e o seu descritor de arquivo, respectivamente. O descritor é utilizado para identificar o programa dentre os demais atualmente carregados no kernel, sendo necessário para interações futuras com esse programa.

```
115 if (bpf_prog_load_xattr(&prog_load_attr, &obj, &prog_fd))
116     return 1;
```

Após o carregamento do programa eBPF no *kernel*, obtém-se a referência para o mapa `rxcnt`. A função `bpf_map__next` retorna um iterador para a lista de mapas declarados no programa. Como nesse caso há apenas um mapa declarado, o valor desse iterador por ser utilizado para se obter o descritor de arquivo referente a ele. A *libbpf.h* também oferece outras funções para se obter descritores de mapa por nome ou por índice na lista de mapas.

```
118 map = bpf_map__next(NULL, obj);
119 if (!map) {
120     printf("finding a map in obj file failed\n");
121     return 1;
122 }
123 map_fd = bpf_map__fd(map);
```

Finalmente, o programa eBPF está pronto para ser carregado no gancho XDP e a aplicação em espaço de usuário pode entrar no laço infinito da função `poll_stats`.

```
130 if (bpf_set_link_xdp_fd(ifindex, prog_fd, xdp_flags) < 0) {
131     printf("link set xdp fd failed\n");
132     return 1;
133 }
134
135 poll_stats(map_fd, 2);
```

A função `poll_stats`, utilizando o descritor de arquivo correspondente ao mapa `rxcnt`, executa consultas periódicas ao mapa e lista todas as entradas existentes, juntamente com as estatísticas calculadas até o momento.

```
35 static void poll_stats(int map_fd, int interval)
36 {
37     unsigned int nr_cpus = bpf_num_possible_cpus();
38     __u64 values[nr_cpus], prev[UINT8_MAX] = { 0 };
39     int i;
40
41     while (1) {
42         __u32 key = UINT32_MAX;
```

```
43
44          sleep(interval);
45
46          while (bpf_map_get_next_key(map_fd, &key, &key) != -1) {
47              __u64 sum = 0;
48
49              assert(bpf_map_lookup_elem(map_fd, &key, values) == 0);
50              for (i = 0; i < nr_cpus; i++)
51                  sum += values[i];
52              if (sum > prev[key])
53                  printf("proto %u: %10llu pkt/s\n",
54                         key, (sum - prev[key]) / interval);
55              prev[key] = sum;
56          }
57      }
58  }
```

Apesar de simples, esse exemplo demonstra algumas questões básicas que devem ser levadas em conta durante o desenvolvimento de programas eBPF, além de mostrar como pode ser feita a interação com programas em espaço de usuário.

3.6. Ferramentas

Esta seção apresentada algumas ferramentas que podem ser de grande utilidade para o desenvolvimento e depuração de programas eBPF.

3.6.1. libbpf

O *kernel* fornece a biblioteca *libbpf* [libbpf 2018], localizada em `tools/lib/bpf`. Ela inclui funções auxiliares para carregar programas e criar e manipular objetos eBPF em espaço de usuário. Para incluir essa biblioteca é necessário informar ao compilador o caminho para o diretório:

```
clang -I <linux-code-dir>/tools/lib ... myprog.c
```

Também é necessário incluir a biblioteca no código C:

```
#include <bpf/libbpf.h>
```

Diversos exemplos disponíveis no diretório `tools/testing/selftests/bpf` demonstram casos de uso da biblioteca, podendo servir como um bom ponto de partida.

3.6.2. iproute2

O *iproute2* é uma coleção de utilitários do espaço de usuário para controlar e monitorar vários aspectos da rede no *kernel* do Linux, incluindo roteamento, interfaces de rede, túneis, controle de tráfego e drivers de dispositivos relacionados à rede. Dentre os produtos mais conhecidos estão as ferramentas `ip` e `tc`. Ambas oferecem maneiras alternativas de carregar códigos eBPF no *kernel*, sem a necessidade de um programa em espaço de

usuário fazendo uso da biblioteca *libbpf*. Com o comando `ip`, por exemplo, é possível carregar códigos no gancho XDP, como mostrado na seção 3.4.5, assim como em níveis mais altos, como no roteamento em camada 3:

```
$ ip route add 192.168.0.0/24 encap bpf headroom 22 xmit
  ↪ obj <bpf-prog> section <section> dev eth0
```

O comando acima adiciona uma regra de roteamento para a subrede 192.168.0.0/24 que executa um programa eBPF localizado na seção `<section>` do arquivo objeto `<bpf-prog>` e encapsula os pacotes com 22 bytes iniciais.

Além disso, o *iproute2* tem a sua própria interface de interação com o sistema eBPF, oferecendo funcionalidades adicionais como a possibilidade de especificar o escopo de alocação de mapas eBPF. Para isso, é necessário declarar um mapa utilizando uma estrutura alternativa (`bpf_elf_map`), definida em `iproute2/include/bpf_elf.h`, e incluir essa estrutura no código C:

```
#include <linux/bpf.h>
#include <iproute2/bpf_elf.h>

struct bpf_elf_map SEC("maps") src_mac = {
    .type = BPF_MAP_TYPE_HASH,
    .size_key = 1,
    .size_value = 6,
    .max_elem = 1,
    .pinning = PIN_GLOBAL_NS,
};
```

Essa estrutura é similar a `bpf_map_def` da *libbpf* porém apresenta campos extras, como o campo `pinning`, utilizado para definir o escopo do mapa, podendo assumir três valores distintos: `PIN_OBJECT_NS`, `PIN_GLOBAL_NS` ou `PIN_NONE`.

Mapas criados com `PIN_OBJECT_NS` têm escopo local, sendo exclusivos do programa que os declarou. Como consequência, mapas com a mesma declaração podem co-existir em programas distintos. Nesse caso, um diretório específico será criado no pseudo-sistema de arquivos BPF para armazenar os nós correspondentes a esses mapas. Caso o valor `PIN_OBJECT_GLOBAL` seja usado, o mapa é criado com um escopo global, habilitando o seu compartilhamento por múltiplos programas. Esse mapa receberá uma entrada no diretório `globals` no pseudo-sistema de arquivos. Por último, o valor `PIN_NONE` indica que o mapa não deve ser fixado no pseudo-sistema de arquivos, impossibilitando a interação com outras aplicações em espaço de usuário.

3.6.3. bpftool

O *bpftool* [bpftool 2018] é uma ferramenta no espaço de usuário essencial para depuração e inserção de programas e mapas BPF. Faz parte da árvore do *kernel* e está disponível em *`tools/bpf/bpftool/`*. Ele pode ser usado para coletar informações sobre programas e mapas do eBPF. Por exemplo, pode-se listar programas carregados:

```
# bpftool prog show
27: xdp tag b722a8b5b9e9be25 dev ens4np0
loaded_at Jun 12/13:20 uid 0
xlated 112B jited 392B memlock 4096B map_ids 31
```

Pode-se imprimir as instruções para este programa com o comando:

```
# bpftool prog dump xlated id 27
0: (b7) r1 = 0
1: (63) *(u32 *)(r10 -4) = r1
2: (bf) r2 = r10
3: (07) r2 += -4
4: (18) r1 = map[id:31]
6: (85) call 0x0#1725914768
7: (b7) r1 = 1
8: (15) if r0 == 0x0 goto pc+3
9: (b7) r1 = 1
10: (c3) lock *(u32 *)(r0 +0) += r1
11: (b7) r1 = 2
12: (bf) r0 = r1
13: (95) exit
```

Além disso, pode-se listar e imprimir o conteúdo de mapas também:

```
# bpftool map
1234: array name ch_rings flags 0x0
key 4B value 4B max_entries 7860 memlock 65536B
# bpftool map dump id 1234
key: 00 00 00 00 value: 00 00 00 00
key: 01 00 00 00 value: 00 00 00 00
key: 02 00 00 00 value: 00 00 00 00
key: 03 00 00 00 value: 00 00 00 00
[...]
Found 7860 elements
```

Também é possível executar algumas operações de gerenciamento, incluindo carregar programas, realizando pesquisas ou atualizações de valores de mapa. Um exemplo para o último item:

```
# bpftool map update id 1234 key 0x01 0x00 0x00 0x00 value 0x12 0x34
  ↪  0x56 0x67
```

3.6.4. llvm-objdump

A ferramenta `llvm-objdump` (versão 4.0 ou superior) pode ser usada para descarregar o código de byte compilado em um formato legível para seres humanos, antes que o usuário tente injetá-lo no *kernel*. Além disso, ela é útil para inspecionar as seções ELF do arquivo eBPF compilado.

```
$ llvm-objdump -S sample_ret0.o
sample_ret0.o: file format ELF64-BPF
```

```
Disassembly of section .text:
func:
; {
0: b7 00 00 00 00 00 00 00 r0 = 0
; return 0;
1: 95 00 00 00 00 00 00 00 exit
```

3.6.5. BPF Compiler Collection (BCC)

O projeto de código aberto *BPF Compiler Collection* (BCC) [BCC 2019a] tem como um de seus objetivos principais facilitar o desenvolvimento de programas baseados em eBPF. Ele oferece um conjunto de *frontends* de compiladores que podem ser usados para interagir com o sistema eBPF utilizando linguagens de alto nível como Python, além de compilar código eBPF diretamente para a linguagem P4 [BCC 2019c].

Além disso, o projeto conta com uma série de ferramentas de exemplo, construídas sobre o BCC, capazes de desempenhar diversas tarefas no sistema operacional. Essas ferramentas podem realizar tarefas como analisar o número de chamadas de sistema executadas por uma aplicação e o tempo decorrido durante uma leitura do disco. Por serem baseadas em programas eBPF são utilizadas por grandes empresas para analisar sistemas de produção reais com baixa sobrecarga adicional.

3.7. Plataformas

Atualmente existem algumas plataformas distintas que utilizam o conjunto de instruções eBPF para adicionar programabilidade em diferentes contextos. Nesta seção são apresentadas as principais delas, divididas em software ou hardware.

3.7.1. Software

A seguir são descritas as plataformas capazes de processar programas eBPF em software, baseadas na implementação original do *kernel* do Linux.

3.7.1.1. *Kernel* do Linux

Como discutido anteriormente, o *kernel* do Linux foi o berço dessa tecnologia e é a plataforma mais popular e com maior desenvolvimento, sendo o foco principal deste texto. Atualmente as aplicações do eBPF no *kernel* já não estão restritas somente à pilha de rede, mas também englobam diversas atividades de instrumentação e monitoramento do sistema operacional. Logo, programas eBPF se tornaram uma importante ferramenta de introspecção utilizada, por exemplo, por projetos de código aberto como IOVisor [IOvisor 2019].

3.7.1.2. Userspace BPF

O projeto de código aberto *ubpf* [ubpf 2019] é uma adaptação da máquina virtual eBPF presente no Linux para o espaço de usuário. Utilizando o interpretador ou o compilador JIT fornecidos por ele é possível embutir uma máquina eBPF em outras aplicações de espaço de usuário, tirando proveito da flexibilidade do conjunto de instruções eBPF.

Muito similar ao *ubpf*, o projeto *rbpf* oferece a mesma funcionalidade porém utilizando a linguagem Rust [Monnet 2019].

Entretanto, diferentemente do sistema eBPF no *kernel*, o *ubpf* não apresenta suporte a mapas e não tem funções auxiliares já implementadas. Apesar disso, ele pode ser estendido para suportar essas operações, como foi feito por Jouet e Pezaros, que o utilizaram como base para o desenvolvimento do comutador BPFabric, discutido a seguir.

3.7.1.3. BPFabric

A arquitetura OpenFlow [McKeown et al. 2008], reconhecida como sendo a implementação *de facto* de SDN, promove uma separação dos planos de controle e de dados das redes e provê uma forma de controlar o *pipeline* de casamento (*match*) do switch utilizando um conjunto limitado de ações e campos. Ela foi criada como um padrão aberto e independente de fornecedor, o que permite que ela seja utilizada em switches em hardware ou software de diferentes fornecedores. Ela apresenta, no entanto, limitações relacionadas ao suporte de novos protocolos e novas ações de casamento. Novas versões do OpenFlow necessitam ser desenvolvidas sempre que inclusões de suporte a casamentos de novos campos ou de novas ações são desejadas.

Para lidar com as limitações do OpenFlow, em [Jouet and Pezaros 2017a] tem-se a proposta de uma nova arquitetura SDN chamada de BPFabric. Essa arquitetura é independente de plataforma, protocolo e linguagem e permite que o plano de dados seja programado a fim de que novas funções possam ser adicionadas ao switch. Tal independência é atingida através da adoção do eBPF como o conjunto de instruções para as funções do plano de dados compiladas.

Como o eBPF não possui conhecimento sobre nenhum protocolo de rede e nem sobre estruturas de pacotes, ele permite que diferentes protocolos possam ser analisados e que um conjunto de tabelas seja utilizado para manter estados e realizar operações de casamento e ação. Desta forma, novas funções podem ser utilizadas para dar suporte a novas funcionalidades e para prover serviços como telemetria, coleta de estatísticas e rastreamento de pacotes.

Para facilitar o desenvolvimento de novas funções, uma linguagem de alto nível restrita é utilizada definir o comportamento do plano de dados. Um exemplo de tal linguagem pode ser um subconjunto restrito de C, onde chamadas de sistemas e algumas bibliotecas externas são excluídas. O código produzido com a linguagem de alto nível é compilado para um arquivo eBFP de execução e ligação (ELF) e deve ser posteriormente carregado no switch pelo controlador. O código compilado define um novo comportamento do switch e reestrutura completamente o seu pipeline de casamento.

Assim como em outras arquiteturas SDN, no BPFabric o switch também é pensado para ser "burro". Nesse sentido, ele não contém nenhuma lógica interna e seu comportamento precisa explicitamente definido pelo controlador. A figura 3.6 mostra uma visão geral da arquitetura do switch, dividida entre os planos de controle e de dados.

O plano de controle foi pensado para ser simples, abrigando apenas um agente que é responsável por intermediar operações entre o controlador e o plano de dados através

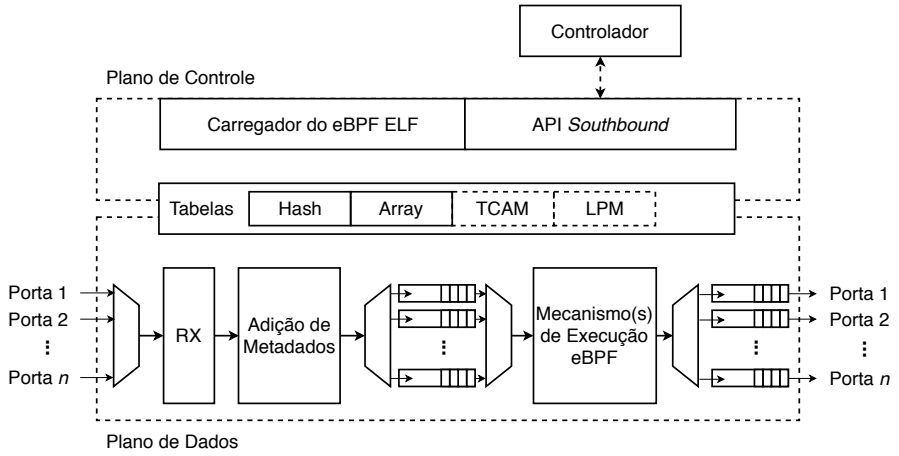


Figura 3.6. Visão simplificada da arquitetura do switch definida no BPFabric.

da API *Southbound*. Entre as operações realizadas pelo agente estão: (i) alterações no comportamento do switch, (ii) recebimento de pacotes e notificações de eventos, e (iii) leituras e atualizações de entradas das tabelas. Quando recebe o código compilado do controlador, o agente faz o uso do Carregador de eBPF ELF para modificar o pipeline do switch. O Carregador é responsável por primeiramente verificar a validade, segurança e o desempenho do programa eBPF a ser carregado. Em caso de sucesso, ele deve realizar a alocação das tabelas eBPF necessárias e converter o código de byte recebido em um formato específico para o switch. Isso faz dele um componente crítico do plano de controle e específico para cada dispositivo.

O plano de dados determina o fluxo descrito a seguir. Os pacotes recebidos pelas interfaces de entrada do switch são armazenados em filas de recebimento com metadados de *timestamp* e informações providas pela camada de enlace sobre eles. Quando um dos mecanismos de execução eBPF se torna disponível, um pacote e seus metadados são retirados de uma das filas de recebimento e então passam pelo pipeline de processamento. É neste estágio que as funções carregadas pelo controlador são executadas. Ao final do pipeline, tem-se o retorno da decisão de encaminhamento do pacote, podendo ela ser o envio dele para o controlador, para alguma porta de saída, para todas as portas de saída (*flood*) ou para nenhuma delas (*drop*).

3.7.2. Hardware

Além das plataformas em software, têm surgido também dispositivos em hardware capazes de executar códigos eBPF. Nesses casos o conjunto de instruções é utilizado para prover programabilidade aos dispositivos e também oferecer uma plataforma alternativa para execução de programas eBPF com maior desempenho.

3.7.2.1. Placa de interface de rede inteligente (*SmartNIC*)

Placas de interface de rede inteligentes (*SmartNICs*) são placas de rede que, além de prover conectividade, permitem que o processamento de tráfego de rede, que seria nor-

malmente feito pela CPU, seja implementado no próprio dispositivo. Elas geralmente proveem funcionalidades de programação e grandes quantidades de memória. Um exemplo de *SmartNIC* é a placa Netronome Agilio CX 2x10GbE [Beckett et al. 2018], que oferece 2 portas 10 GbE emparelhadas com um processador ARM11, 2 GB de memória RAM DDR3 e mais de 100 núcleos de processamento dedicado.

Devido à grande capacidade de processamento das *SmartNICs* atuais, elas têm se tornado um grande atrativo para o eBPF. O descarregamento do eBPF em *hardware* têm a capacidade de propiciar diversas vantagens e ganhos em desempenho. A redução da latência é uma das vantagens possíveis, visto que pacotes não precisam deixar a placa para serem processados. Outra vantagem é a habilidade de carregar programas *on-the-fly* que o eBPF propicia, permitindo a troca dinâmica de programas em um *data center* operante sem a necessidade de reinicialização dos sistemas. Além disso, a programação de *SmartNICs* através do eBPF facilita a implementação de recursos como limitadores de fluxos e filtragem de pacotes, o que permite o desenvolvimento de aplicações eficientes para a mitigação de ataques de negação de serviço (*Denial of Service* – DoS), balanceamento de cargas, comutação de pacotes, dentre outras.

3.7.2.2. Roteador eBPF em Hardware

Processamento de pacotes sobre hardwares com plano de dados programável tornaram-se um campo de pesquisa ativo em redes de computadores que ganhou atenção da indústria e academia da área [Dargahi et al. 2017]. Hardwares com essa característica possibilitam o processamento de pacotes de modo flexível e expressivo sem conhecer comandos e especificações de baixo nível do plano de dados via espaço do usuário. Usuários ao utilizar esse tipo de hardware podem analisar, processar e encaminhar pacotes utilizando tabelas de casamento e ações de modo dinâmico. Atualmente, hardwares como ASICs, CPUs x86, GPUs e FPGAs estão sendo utilizados no processamento de pacotes de redes *Gigabit Ethernet* [Sharma et al. 2017].

FPGAs comparado a outras plataformas de hardware tornaram-se mais atrativos para área de redes de computadores por combinar alto poder de processamento com flexibilidade. Empresas como Microsoft, Baidu e Amazon já inseriram FPGAs em seus provedores de nuvem para acelerar o processamento de pacotes de suas redes. FPGA é um dispositivo composto por blocos lógicos programáveis, memórias e dispositivos de entrada e saída. A programação neste equipamento ocorre utilizando linguagens de descrição de hardware (LDHs), por exemplo, Verilog ou VHDL [Wang et al. 2017].

Aplicações de rede são implementadas sobre FPGAs sintetizando programas descritos usando LDHs, e carregando o arquivo *Bitstream* gerado após a síntese. Para carregar um arquivo *Bitstream* o usuário pára o funcionamento do equipamento. Além disso, uma síntese pode levar horas e cada modificação realizada no circuito acarreta uma nova síntese. Usuários gastam horas simulando e reescrevendo circuitos que não funcionam após síntese. Todas essas dificuldades em conjunto tornam o processo de desenvolvimento de aplicações de rede uma tarefa complexa e desafiadora que demanda muito esforço e conhecimento do hardware por parte do usuário [Zilberman et al. 2015].

Para lidar com as limitações do OpenFlow e FPGAs, [Pacífico et al. 2018] propôs

uma arquitetura de roteador eBPF em hardware que processa pacotes independente de protocolo, e utiliza instruções eBPF geradas a partir de programas C ou P4 criados pelo usuário para definir como os pacotes serão processados no plano de dados. A principal característica deste roteador é permitir que usuários em tempo de execução possam definir dinamicamente a utilização de novos campos e protocolos, sem recompilar ou reiniciar o roteador. O roteador proposto foi implementado sobre a plataforma da NetFPGA 1G, utilizando o poder de processamento de um hardware reconfigurável para processar pacotes dinamicamente de redes 1G.

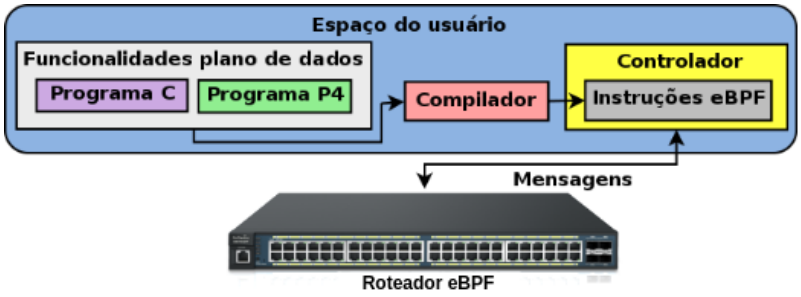


Figura 3.7. Visão geral do roteador eBPF em hardware [Pacífico et al. 2018].

A arquitetura do roteador eBPF (figura 3.7) é composta por um plano de controle (espaço do usuário) e um plano de dados. O plano de controle é formado por um controlador centralizado que envia e instala programas eBPF de aplicações de rede desenvolvidas pelo usuário. Programas eBPF são enviados via conexão *socket* estabelecida entre espaço de usuário e plano de dados. Programas eBPF são compilados e verificados em tempo de execução, e antes de serem instalados no roteador são checados pelo verificador eBPF no espaço do usuário.

O plano de dados do roteador eBPF contém um processador eBPF que realiza operações de análise, casamento e ações dinâmicas de pacotes independente de protocolo utilizando instruções eBPF. O processamento de pacotes no plano de dados do roteador começa quando as instruções eBPF são carregadas na memória de instruções do processador.

A figura 3.8 mostra o caminho de dados do roteador. Os módulos na cor laranja representam os módulos do roteador. Os módulos na cor cinza são os módulos padrão da NetFPGA 1G. À medida que um pacote chega na fila FIFO do módulo *Output Port Lookup*, a (*Finite State Machine* - FSM) retira o pacote da fila e armazena o pacote na memória de dados do processador. Em seguida, o contador de programa (PC) do processador será inicializado, executando as instruções eBPF instaladas. Quando a última instrução for executada o valor do registrador r0 será enviado para o módulo "Ação no pacote". Este módulo é responsável por definir a ação que será realizada no pacote (encaminhamento, descarte ou inundação) quando o pacote for enviado para o módulo *Output Queues*. Após a ação definida, o pacote será retirado da memória de dados, e enviado para *Output Queues*. Em seguida, a FSM espera o próximo pacote para recomeçar o processamento.

Quando PC é inicializado as instruções são buscadas na memória de instruções

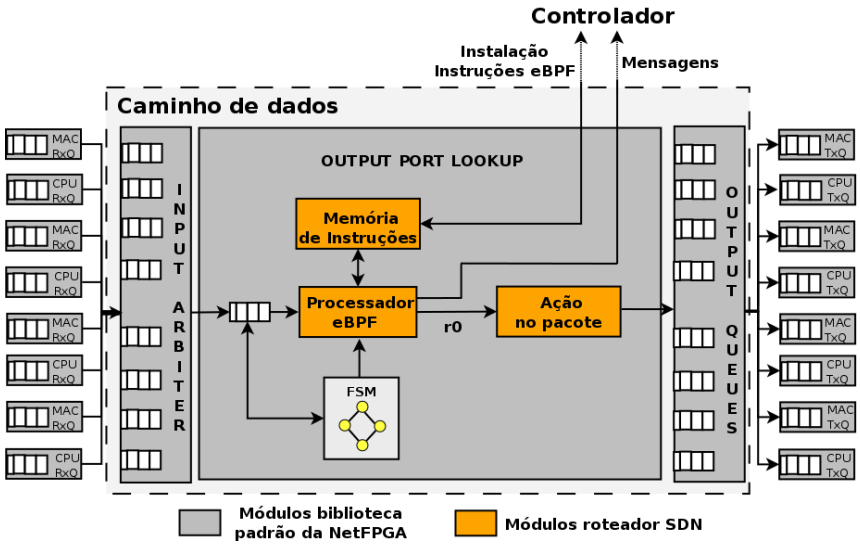


Figura 3.8. Caminho de dados do roteador eBPF implementado na NetFPGA 1G [Pacífico et al. 2018].

de acordo com o endereço atual do PC. Em seguida, as instruções são decodificadas e o valor dos registradores origem e destino são lidos. O próximo passo é executar a operação da instrução na ULA que pode retornar um valor ou um endereço para acesso a memória de dados. Todas as instruções terminam de ser executadas quando ocorre uma escrita na memória de dados ou uma escrita banco de registradores, exceto as instruções de salto (*jump*) que terminam após o retorno da saída da ULA. A figura 3.9 mostra o caminho de dados e controle do processador eBPF.

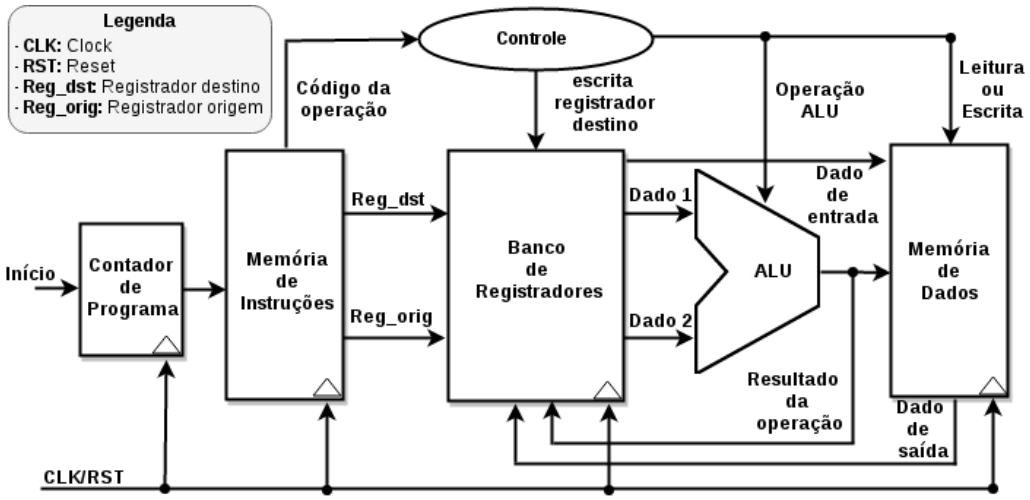


Figura 3.9. Caminho de dados e controle do processador eBPF [Pacífico et al. 2018].

Todos os experimentos foram realizados em um ambiente real composto por dois terminais (A e B) e um controlador conectados no roteador. Os autores validaram o roteador medindo vazão por tamanho de pacote, vazão após reinstalar um programa e vazão trocando entre dois programas distintos (encaminhamento e lista de controle de acesso). Os programas eBPF foram escritos na linguagem C e compilados usando LLVM 3.9 para

plataforma eBPF. A ferramenta *objcopy* foi utilizada para extrair o seguimento no arquivo `elf` referente as instruções. O controlador foi responsável por instalar as instruções no roteador. Em todos os experimentos a ferramenta *iperf* foi usada para criar conexões TCP e UDP entre os terminais. Os resultados obtidos demonstraram que o roteador eBPF tem um plano de dados que permite análise, casamento e ações dinâmicas.

As principais limitações encontrados na implementação do roteador eBPF foram a quantidade de recurso disponível (células lógicas) da NetFPGA 1G e sintetizar instruções como multiplicação, divisão e resto da divisão. Essas instruções são suportadas pelo eBPF, no entanto, a ferramenta da *Xilinx* da NetFPGA 1G não é capaz de sintetizar tais instruções. Os autores descreveram que uma FPGA com quantidade maior de recursos e uma versão de sintetizador recente, por exemplo, NetFPGA SUME usando *Xilinx Vivado*, contornam essas limitações de implementação.

3.8. Funções de rede

Nesta seção são apresentadas algumas funções de redes que demonstram a utilidade e a flexibilidade do eBPF em diferentes plataformas.

3.8.1. Balanceador de carga L4 - l4lb

A Netronome disponibiliza o código de um programa XDP chamado l4lb que implementa um balanceador de carga L4 [Netronome 2019]. O programa l4lb processa os pacotes da rede e calcula um valor hash com base no endereço IP origem, juntamente com as portas TCP ou UDP. O hash gerado é usado como chave em um mapa eBPF.

O mapa eBPF é preenchido com o endereço dos servidores disponíveis para os quais o programa l4lb pode redirecionar o pacote. Este programa estende e insere um cabeçalho IP externo com os dados do mapa. Após o processamento, o programa envia o pacote para o servidor de destino. A computação deste programa pode ser toda realizada pela placa de rede, poupando ciclos para a CPU.

Os scripts para preencher mapas e ler estatísticas são escritos em Python. Esses scripts utilizam o *bpftool* para mostrar como um programa pode ser implementado usando os utilitários *iproute2* e *bpftool*.

3.8.2. RSS Programável

É um escalonador programável de pacotes recebidos para múltiplas CPUs. A técnica de *Receive Side Scaling* (RSS) [Netronome 2019] é utilizada por muitas placas de rede para distribuir a computação dos pacotes para um conjunto de CPUs distintas por meio do uso de múltiplas filas. Porém as implementações geralmente são proprietárias e feitas em hardware, permitindo pouca ou nenhuma programabilidade. Utilizando um programa eBPF, a distribuição dos pacotes pode ser modificada sob demanda por meio de valores associados a mapas ou da substituição completa do programa eBPF carregado.

3.8.3. Monitoramento

Para demonstrar a flexibilidade do BPFabric, os autores implementaram diversas funções de rede de exemplo, disponíveis no repositório oficial do projeto [Jouet and Pezaros 2017b].

Dentre os exemplos estão funções de monitoramento de rede em tempo real. Por se tratar de um dispositivo programável, diferentes tipos de medições podem ser implementadas e instaladas sob demanda no comutador juntamente com a funcionalidade padrão de encaminhamento. Os tipos de medição incluem análise do tempo de chegada entre pacotes, latência média, distribuição do tamanho dos pacotes recebidos e detecção de anomalias.

Por exemplo, a aplicação de análise de tempo de chegada entre pacotes utiliza um mapa do tipo *array* de tamanho 1 para armazenar informações de estado do fluxo de pacotes (tempo de chegada do último pacote, contador de pacotes observados e quantidade de intervalos maiores que o máximo definido) e outro, também do tipo *array*, para armazenar a quantidade de pacotes que chegaram após certos intervalos de tempo de seus predecessores. Quando um pacote chega ao switch, busca-se as informações de estado do fluxo armazenadas no mapa e então calcula-se o intervalo de tempo decorrido desde a chegada do último pacote observado anteriormente. Caso esse tempo seja maior do que o maior intervalo disponível no *array*, então o contador de quantidade de intervalos maiores que o máximo definido é incrementado. Caso contrário, verifica-se qual dos tamanhos de intervalo disponíveis no segundo *array* é o mais próximo do medido e se incrementa e atualiza o valor do contador correspondente. Depois são atualizadas as informações de último pacote recebido e o contador de pacotes observados.

3.8.4. NAT

É possível implementar um tradutor de endereços de rede (NAT) utilizando um programa eBPF e o roteador citado na seção 3.7.2.2. Essa função sobrescreve o endereço IP e porta L4 de origem dos pacotes enviados pelos hospedeiros da rede com um novo par de valores. O mapeamento entre o par original e o novo deve ser armazenado em uma tabela local para posterior consulta. Quando um pacote vindo da Internet é recebido, essa tabela é consultada, utilizando o par *(IP destino, porta destino)* do pacote como chave, e os valores originais são reescritos no pacote. Dessa forma, a rede é capaz de reenviar o pacote para o hospedeiro correspondente. A tabela pode ser implementada utilizando um mapa do tipo `BPF_MAP_TYPE_HASH`, e as operações sobre ela são feitas com as funções auxiliares `bpf_map_lookup` e `bpf_map_update`.

3.8.5. Filtragem de aplicações

Uma operação muito comum feita por administradores de redes é restringir acesso a aplicações de rede específicas. Isso pode ser feito através da filtragem de pacotes com determinados valores de porta de destino nos cabeçalhos de camada de transporte. Por meio do acesso direto ao pacote, um programa eBPF pode processar os cabeçalhos e extrair a porta de destino utilizada, podendo tomar a decisão de efetuar o descarte ou não. Um exemplo dessa aplicação está disponível no repositório deste minicurso [Vieira et al. 2019].

3.8.6. ChaCha

Também é possível implementar uma versão do algoritmo de criptografia ChaCha20 [Nir and Langley 2018] no gancho XDP. Esse algoritmo é utilizado por empresas como o Google [Google 2014] para criptografia simétrica no protocolo *Transport Layer Security* (TLS). Um exemplo de implementação está disponível no repositório [Vieira et al. 2019]. Por conta da limitação do tamanho do código eBPF permitido pelo Linux, o algoritmo

implementado utiliza apenas 8 rodadas, ao invés dos 20 tradicionais.

3.9. Projetos de pesquisa

Nesta seção são descritos projetos de pesquisa que utilizam eBPF para o processamento de pacotes sobre plataformas de software.

3.9.1. Encadeamento de funções de serviço

O encadeamento de funções é um problema recorrente em ambientes de virtualização de funções de redes (*Network Function Virtualization* - NFV). Para oferecer serviços complexos, muitas vezes é necessário combinar diferentes funções (NAT, *proxy*, *firewall*, DPI, etc) em uma ordem pré-determinada, também denominada cadeia. Pacotes de diferentes fluxos podem ser processados por cadeias diferentes com funções e tamanhos distintos, exigindo flexibilidade além da oferecida por mecanismos de roteamento e encaminhamento tradicionais. A *Internet Engineering Task Force* (IETF) propôs uma arquitetura de referência com esse fim [Halpern and Pignataro 2015]. Esse padrão define diferentes elementos de rede que atuam em conjunto sobre um protocolo de encapsulamento especial para habilitar a movimentação dos pacotes pelas funções de uma cadeia na ordem correta. A RFC 8300 [Quinn et al. 2018] define o protocolo *Network Service Header* (NSH), que pode ser usado para esse fim. De forma geral, a implementação desses elementos é feita de forma indireta, através da inclusão de suas ações em dispositivos já existentes, como roteadores e comutadores de rede. Porém, isso cria um alto acoplamento entre a arquitetura de encadeamento e a infraestrutura de rede.

Castanho et al. [Castanho et al. 2019] propõem uma nova arquitetura que visa desacoplar o encadeamento dos dispositivos de rede, denominada *Cadeia-Aberta*. Nela os elementos da RFC 7665 são integrados às funções de rede como estágios de processamento. Cada função de rede é executada por uma máquina virtual ou contêiner individual, na qual os estágios são carregados na forma de programas eBPF. Toda a funcionalidade necessária para habilitar o encadeamento é implementada pelos estágios. Com isso, não há mais a necessidade de elementos intermediários entre as funções para executar o encadeamento, tornando a arquitetura transparente tanto à rede quanto às funções, não exigindo nenhuma modificação nas mesmas e facilitando a sua adoção em ambientes legados.

O processamento é dividido em três estágios: *Dec*, *Enc* e *Fwd*, como mostrado na figura 3.10. Assim que um quadro é recebido, o estágio *Dec* verifica se ele está encapsulado com um cabeçalho NSH. Nesse caso, o encapsulamento de transporte e o NSH são removidos e o pacote é enviado para o espaço de usuário para ser processado pela função de rede (SF). Esse estágio foi implementado no gancho XDP para ter acesso ao pacote no nível mais baixo possível da pilha do *kernel*. Para efetuar o desencapsulamento foi utilizada a função auxiliar `bpf_xdp_adjust_head`:

```
// Remove outer encapsulation + NSH
bpf_xdp_adjust_head(ctx, (int)(outer_header_size + sizeof(struct
  ↪  nshhdr)));
```

As informações do NSH são salvas em um mapa BPF para serem readicionadas ao

pacote durante a transmissão, já no estágio *Enc*. De forma similar ao estágio *Dec*, os estágios *Enc* e *Fwd* precisam modificar o pacote no nível mais baixo possível da rede. Como o gancho XDP está disponível apenas na recepção, o gancho TC foi utilizado para esses dois últimos estágios. No estágio *Enc*, o cabeçalho NSH original é readicionado ao pacote, que é então encaminhado para o estágio *Fwd*. A readição do NSH é possível pois os estágios *Dec* e *Enc* compartilham um mapa de armazenamento temporário de cabeçalhos NSH. Porém, diferentemente do XDP, o gancho TC não apresenta uma função de encapsulamento genérica. Por conta disso os autores utilizaram múltiplas chamadas à função `bpf_skb_vlan_push` para inserir rótulos VLAN que pudessem ser sobrescritos com o encapsulamento desejado:

```
#pragma clang loop unroll(full)
for(int i = 0 ; i < 8 ; i++){
    ret = bpf_skb_vlan_push(...);
    if (ret < 0) {
        return TC_ACT_SHOT;
    }
}
```

Por fim, o estágio *Fwd* utiliza as informações do cabeçalho NSH recém-adicionado para definir a próxima função a ser executada na cadeia. Após uma consulta em um mapa local, o encapsulamento de transporte é alterado com o endereçamento correspondente para que a rede possa encaminhar o pacote para o próximo salto.

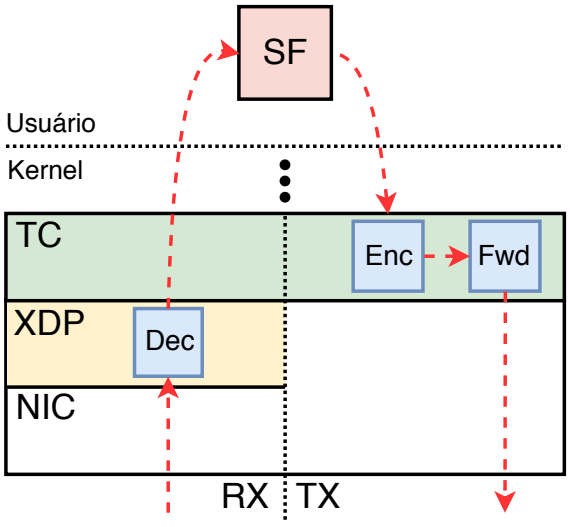


Figura 3.10. Estágios de processamento da arquitetura Cadeia-Aberta implementados como códigos eBPF no *kernel* do Linux [Castanho et al. 2019].

3.9.2. Roteamento por segmento

A demanda recente por maior programabilidade de redes de computadores têm levado ao surgimento de tecnologias como o roteamento por segmento [Filsfils et al. 2018]. Essa técnica permite que administradores de redes especifiquem diferentes ações a serem executadas sobre pacotes em pontos específicos da rede. Utilizando rótulos MPLS ou o protocolo IPv6 com um campo especial chamado *Source Routing Header* (SRH), cada pacote é

encapsulado com uma lista ordenada de ações de roteamento e processamento, denominadas segmentos. Conforme o pacote é movimentado pela rede, dispositivos habilitados processam essa a lista de segmentos e executam as ações especificadas por ela. Isso permite a implementação, por exemplo, de diferentes funções de rede [Abdelsalam et al. 2017].

O *kernel* do Linux já apresenta suporte a roteamento por segmento sobre IPv6 desde a versão 4.10, porém com apenas poucas opções de processamento, como encaminhamento e envio e recebimento de pacotes rotulados. [Duchene et al. 2018] utilizaram o arcabouço eBPF para tornar a criação e especificação de novos segmentos mais genérica e flexível. Por meio da adição de novas funções auxiliares, os autores extenderam o *kernel* do Linux para permitir a implementação de segmentos na forma de programas eBPF. Com isso, novas funções de rede podem ser facilmente desenvolvidas e associadas a regras de roteamento do Linux, facilitando sua integração a ambientes de roteamento por segmento sobre IPv6.

3.9.3. Sketches

Sketches são estruturas de dados probabilísticas compactas que são utilizadas por algoritmos de *streaming* para manter informações resumidas sobre fluxos [Yu et al. 2013]. Suas principais vantagens são: (i) utilização de memória para armazenamento de informações significativamente menor que o volume de dados de entrada, e (ii) *tradeoffs* provados entre a quantidade de memória utilizada e a acurácia obtida. Santos *et al.* [Santos et al. 2019b] propõem e avaliam implementações de aplicações de monitoramento de redes no BPFabric que utilizam *sketches* implementados utilizando eBPF.

A seguir, são apresentados alguns dos *sketches* que podem ser utilizados em diferentes tarefas de monitoramento de rede. Após, descreve-se o que é necessário para implementar *sketches* utilizando o eBPF através do BPFabric.

3.9.3.1. Bloom Filter

O *Bloom Filter* [Bloom 1970] é uma estrutura probabilística que é utilizada para verificar se um elemento pertence ou não a um determinado conjunto. Eficiente em termos de espaço, ela é formada por um *array B* de m bits. Ela também utiliza k funções hash $h_1, h_2, ..., h_k$ para mapear um elemento e para k posições do *array B*. A figura 3.11 apresenta essa estrutura probabilística.

Inicialmente iguais a 0, os bits de B são ativados (se tornam iguais a 1) à medida que novos elementos são inseridos ao conjunto. A inserção de um novo elemento e é feita através da ativação dos bits nas posições $h_1(e), h_2(e), ..., h_k(e)$ de B, ou seja, fazendo-se $B[h_i(e)] = 1$ para $i = 1, 2, ..., k$. Já para determinar se um elemento e_x pertence ao conjunto, é necessário verificar se todos os bits $B[h_i(e_x)]$, para $i = 1, 2, ..., k$, estão ativados. Caso um bit não esteja ativado, este elemento não está no conjunto.

É interessante notar que o Bloom Filter não apresenta falsos negativos, ou seja, ele não acusa a existência de um elemento no conjunto que realmente não faça parte dele. Entretanto, falsos positivos são possíveis. Também destaca-se que a implementação de um Bloom Filter com uma dada uma tolerância a erros, expressa como uma probabilidade p,

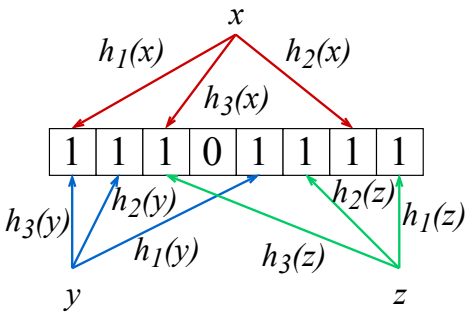


Figura 3.11. Bloom Filter: Elementos *x*, *y* e *z* são mapeados para, por exemplo, 3 posições cada, que indicam quais bits devem ser ativados/verificados.

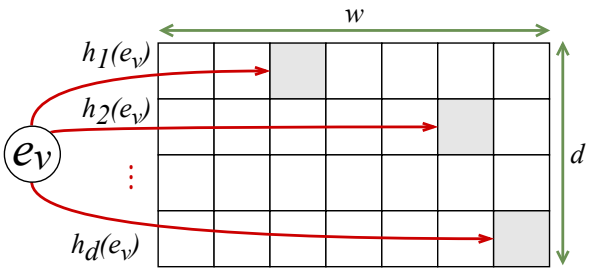


Figura 3.12. Count-Min Sketch: Entrada *e* é mapeada para *d* contadores utilizando *d* funções hash.

e com um número esperado de elementos no conjunto igual a n, requer a utilização de $k = -\log_2 p$ funções hash e de um *array* de tamanho $m = -(n \ln p)/(\ln 2)^2$ bits.

3.9.3.2. Count-Min Sketch

O *Count-Min Sketch* [Cormode and Muthukrishnan 2005] é uma estrutura de dados probabilística compacta que é utilizada para contar a frequência de eventos. Ela é composta por uma matriz M, de tamanho $w \times d$, de contadores que são inicializados com o valor zero. Ela também utiliza um conjunto de funções hash $h_1, h_2, ..., h_d$ independentes par-a-par para mapear um elemento de entrada a um contador em cada uma das linhas da matriz. A figura 3.12 mostra uma representação dessa estrutura de dados.

Existem dois procedimentos fundamentais no Count-Min Sketch: a atualização e a estimativa. O procedimento de atualização registra o acontecimento de um evento e_v através do incremento dos contadores nas posições $M[1,h_1(e_v)], M[2,h_2(e_v)], ..., M[d,h_d(e_v)]$. Já o procedimento de estimativa é utilizado para estimar a frequência de ocorrência do evento e_v como $\hat{f}_e = \min_{i=1,...,d} M[i,h_i(e_v)]$. Observa-se que $\hat{f}_e \geq f_e$, onde f_e é a frequência real do evento e_v. Se $w = \lceil e/\varepsilon \rceil$ e $d = \lceil \ln(1/\delta) \rceil$, então tem-se com probabilidade $1-\delta$ que $\hat{f}_e \leq f_e + \varepsilon N$, onde e é o número de Euler e N é o número total de eventos registrados no sketch.

3.9.3.3. K-ary Sketch

O *K-ary Sketch* é uma estrutura de dados probabilística projetada para estimar a atividade total de uma dada chave em um fluxo [Krishnamurthy et al. 2003]. Similar ao Count-Min Sketch, ela consiste de uma matriz H com dimensões $w \times d$, onde cada linha i é associada a uma função hash h_i. As funções hash utilizadas devem ser 4-universais para que a precisão das estimativas tenham garantias probabilísticas. A figura 3.13 apresenta uma representação dessa estrutura.

Essa estrutura oferece operações para atualizar o *sketch*, estimar um valor atra-

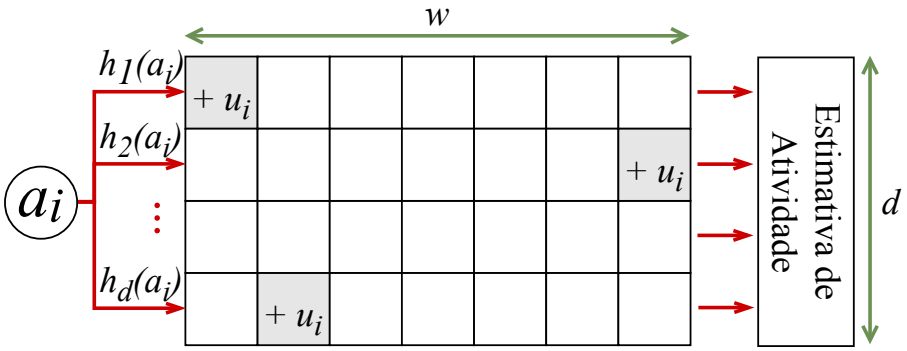


Figura 3.13. K-ary Sketch: Computa-se uma função hash para cada uma das *d* linhas e os contadores associados são incrementados.

vés de uma chave e computar a combinação linear de múltiplos *sketches*. Considerando um fluxo de dados $S = \alpha_1, \alpha_2, ...$, onde $\alpha_i = (a_i, u_i)$ é um par de chave (a_i) e valor de atualização (u_i), a operação de atualização de um item no sketch é dada por $\forall j \in \{1,...,d\}, H[j][h_j(a_i)] = H[j][h_j(a_i)] + u_i$. Já a operação de estimativa para uma chave a_i computa o seguinte valor:

$$mediana_{j \in \{1,...,d\}} \left\{ \frac{H[j][h_j(a_i)] - soma(S)/w}{1 - 1/w} \right\}, \tag{1}$$

onde $soma(S) = \sum_{k \in \{1,...,w\}} H[0][k]$ só precisa ser computado uma vez antes de qualquer estimativa de valores. A operação de combinação une múltiplos sketches fazendo $S = \sum_{l=1}^{n} c_l S_l$, onde c_l é um peso associado ao sketch S_l. Para fazer isto, ela combina cada entrada da matriz H da seguinte forma $H[j][k] = \sum_{l=1}^{n} c_l \times H_l[j][k]$. Considerando esta forma de combiná-los, a estimativa provida pelo k-ary Sketch é estatisticamente sem viés.

3.9.3.4. PCSA

A *Contagem Probabilística com Média Estocástica* (PCSA, do inglês *Probabilistic Counting with Stochastic Averaging*) [Flajolet and Martin 1985] foi proposta com o intuito de estimar o número de elementos distintos em uma grande coleção. Para se ter uma ideia, o PCSA é capaz de contar bilhões de elementos utilizando apenas um número comparavelmente pequeno de bits.

A inserção de um novo elemento ao PCSA utiliza uma função hash e outras duas funções $r(x)$ and $R(x)$. Aqui considera-se que a função hash retorna um inteiro (de 32 ou 64 bits). A função $r(x)$ conta o número de 1's à direita no número de entrada x. Já a função $R(x)$ é definida como $R(x) = 2^{r(x)}$ e é computada utilizando as instruções $\neg x \ \& \ (x+1)$.

O PCSA armazena uma matriz de tamanho $w \times d$, onde w pode ser igual a 32 ou 64 bits e d, o número de linhas na matriz, é utilizado para controlar a precisão dessa estrutura. Quando um novo elemento é adicionado, a função hash é computada e retorna um número k que indica qual a linha da matriz deverá receber o novo dado. Dada a linha k selecionada, executa-se a operação $matriz[k] = matriz[k] \mid R(x)$.

A contagem do número de elementos no PCSA requer a computação da soma $s = \sum_{i=1}^{d} r(matriz[i])$. Esse valor é utilizado para obter o resultado da contagem, que é dado por $d * 2^{(s/d)/.77351}$. A precisão do PCSA, dada uma matriz de $d \times 64$ bits, é igual a $0.78\sqrt{d}$. Esta operação é apresentada no Algoritmo 1.

Algoritmo 1 Operação de contagem do PCSA

```
1: procedure CONTAELEMENTOS(Matriz)
2:     soma = 0.0
3:     for i = 0; i < tamanho(Matriz) do
4:         soma+ = r(Matriz[i])
5:     end for
6:     media = soma/tamanho(Matriz)
       return tamanho(Matriz) * 2^(media/.77351)
7: end procedure
```

3.9.3.5. Implementação de sketches utilizando eBPF

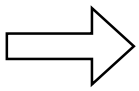
É possível implementar sketches para o eBPF no BPFabric a partir da definição de novos tipos de mapas (seção 3.5.2). De forma geral, para criar um novo mapa é necessário (i) definir um novo tipo que será utilizado para definir os parâmetros do novo mapa, (ii) a nova estrutura de dados que comporta essa estrutura, e (iii) definir as operações desse novo tipo. Exemplos de implementações dos sketches mencionados anteriormente podem ser obtidos em [Santos et al. 2019a].

A definição de um novo tipo de mapa é feita através da inserção de uma nova entrada no *enum* `bpf_map_type`, localizada no arquivo *bpfmap.h*. Por exemplo, a definição de um novo tipo correspondente ao Bloom Filter pode ser feita através da adição do identificador `BPF_MAP_TYPE_BITMAP` ao final do `bpf_map_type`:

```
enum bpf_map_type {
    BPF_MAP_TYPE_UNSPEC,
    BPF_MAP_TYPE_HASH,
    BPF_MAP_TYPE_ARRAY,
    BPF_MAP_TYPE_BITMAP,
};
```

Além disso, como apresentado na seção 3.5.2, os parâmetros de criação de uma instância de um mapa são definidos pela *struct* de configuração `bpf_map_def`. Essa *struct* contém atributos que definem o tipo do mapa (*map_type*), tamanho da chave, tamanho dos valores armazenados, dentre outros necessários para a criação. Como cada sketch possui um conjunto de parâmetros diferentes, é possível adicionar ou modificar campos da `bpf_map_def` para que ela comporte parâmetros específicos. No BPFabric, isso pode ser feito editando o arquivo *includes/ebpf_switch.h*. Seguindo com o exemplo do Bloom Filter, seria possível modificar a *struct* para adicionar parâmetros como a quantidade de funções hash (k) e a quantidade de bits (m) do *array*. Uma forma de fazer isso é produzir a seguinte modificação:

```
struct bpf_map_def {
    unsigned int type;
    unsigned int key_size;
    unsigned int value_size;
    unsigned int max_entries;
    unsigned int map_flags;
};
```



```
struct bpf_map_def {
    unsigned int type;
    union{
        unsigned int key_size;
        unsigned int num_hashes;
    };
    union{
        unsigned int value_size;
        unsigned int num_bits;
    };
    unsigned int max_entries;
    unsigned int map_flags;
};
```

Para definir as operações de um novo tipo é necessário implementar um conjunto de operações básicas definidas no BPFabric. Algumas delas são: `map_alloc`, responsável pela criação do mapa e alocação de memória; `map_free` responsável por liberar a memória utilizada no mapa; `map_lookup_elem`, utilizado para pesquisar um o valor associado a uma chave em um mapa; `map_get_next_key`, utilizado para retornar a próxima chave de um mapa; `map_update_elem`, utilizado para atualizar o valor associado a uma chave; `map_delete_elem`, utilizado para deletar o valor associado a uma chave. A implementação dessas operações deve ocorrer em um par de arquivos .c e .h (por exemplo, *bitmap.c* e *bitmap.h*) e as assinaturas das funções implementadas devem ser correspondidas às referências do BPFabric utilizando o vetor de operações de mapas `bpf_map_types`, disponível no arquivo *bpfmap.c*. Vale notar que também é necessário incluir nesse arquivo uma referência para o arquivo `header` com as definições das operações do novo tipo.

Considerando o exemplo de implementação de um Bloom Filter, pode-se definir as seguintes funções: `bitmap_map_alloc`, que fica responsável por criar e inicializar a matriz que será utilizada pelo bitmap; `bitmap_map_free` que libera a memória alocada pelo bitmap; `bitmap_map_lookup_elem` que recebe uma chave e retorna uma variável indicando se a chave se encontra no sketch utilizando as operações apresentadas na seção 3.9.3.1; `bitmap_map_update_elem` que recebe uma chave e uma flag, sendo a que flag possibilita escolher se a função deve limpar o bitmap ou adicionar um novo elemento a ele. Funções como `bitmap_map_delete_elem` e `bitmap_map_get_next_key` podem ser definidas mas não precisam ser implementadas, já que não existem operações no sketch que possam ser associadas a essas funções. Um exemplo de como ficaria o vetor `bpf_map_types` após a adição das referências das operações implementadas Bloom Filter (identificadas como `bitmap_*`) pode ser visto abaixo.

```
const struct bpf_map_ops bpf_map_types[] = {
    [BPF_MAP_TYPE_HASH] = {
        .map_alloc = htab_map_alloc,
        .map_free = htab_map_free,
```

```
        .map_get_next_key = htab_map_get_next_key,
        .map_lookup_elem = htab_map_lookup_elem,
        .map_update_elem = htab_map_update_elem,
        .map_delete_elem = htab_map_delete_elem,
    },
    [BPF_MAP_TYPE_ARRAY] = {
        .map_alloc = array_map_alloc,
        .map_free = array_map_free,
        .map_get_next_key = array_map_get_next_key,
        .map_lookup_elem = array_map_lookup_elem,
        .map_update_elem = array_map_update_elem,
        .map_delete_elem = array_map_delete_elem,
    },
    [BPF_MAP_TYPE_BITMAP] = {
        .map_alloc = bitmap_map_alloc,
        .map_free = bitmap_map_free,
        .map_get_next_key = bitmap_map_get_next_key,
        .map_lookup_elem = bitmap_map_lookup_elem,
        .map_update_elem = bitmap_map_update_elem,
        .map_delete_elem = bitmap_map_delete_elem,
    }
};
```

Existem duas considerações que podem ser feitas ainda na parte de definição de operações. A primeira delas é que é possível definir novas operações para os mapas. Se existe alguma operação no sketch que não corresponde a uma operações já definidas, uma nova operação pode ser adicionada. Essa adição pode ser feita através da inclusão de um novo campo na estrutura de operações `bpf_map_ops`, disponível no arquivo bpfmap.h. A segunda consideração é que nem todas as operações definidas necessitam ser implementadas para todos os mapas. Por exemplo, a operação *map_get_next_key* não tem uma função para alguns sketches, podendo assim não ser implementada.

3.9.4. Outros projetos

Além dos já citados, há vários outros projetos que fazem uso do arcabouço eBPF para adicionar programabilidade ao plano de dados em diferentes contextos. [Ahmed et al. 2016] utilizam programas eBPF para modificar o caminho de dados de redes virtuais de centros de dados. [Jouet et al. 2015] e [Tu et al. 2017] propõem extensões ao protocolo OpenFlow e ao comutador virtual Open vSwitch, respectivamente, utilizando programas eBPF. [Bertrone et al. 2018] propõem uma nova versão da ferramenta *iptables* utilizando a tecnologia.

Em cenários de IoT e no paradigma de computação de borda, as mesmas informações de um sensor podem ser usadas por vários aplicativos em locais diferentes. Neste caso, o fluxo de dados precisa ser replicado. Em [Baidya et al. 2018], o fluxo de tráfego e replicação de pacotes para cada processo de computação específico é controlado por um programa eBPF.

Algumas empresas já utilizam eBPF em ambientes de produção na indústria, como é o caso da Cloudflare, que utiliza XDP em seu sistema de mitigação de ataques de negação de serviço [Bertin 2017], e do Facebook, que desenvolveu um balanceador de carga baseado em eBPF chamado Katran [Facebook 2018].

Por fim, projetos de código aberto baseados em eBPF também tem surgido nos últimos anos. Cilium [Cilium 2018] é um projeto para prover segurança em redes de contêineres e aplicações com microserviços. O projeto IOVisor [IOvisor 2019] mantém múltiplos sub-projetos baseados em eBPF como *gobpf* [gobpf 2019], que permite a interação com o sistema eBPF utilizando a linguagem Go, as ferramentas *ply* [ply 2019] e *bpftrace* [bpftrace 2019] para introspecção do kernel, além do BCC [BCC 2019a] e do *ubpf* [ubpf 2019], já discutidos anteriormente.

3.10. Desafios e limitações

A tecnologia eBPF habilita flexibilidade no processamento de pacotes por permitir a execução de códigos de modo dinâmico sobre o *kernel* sem instalar qualquer módulo extra. Essa tecnologia também suporta cadeia de serviços arbitrários (conjunto de funções de rede conectadas) e integração com XDP para ter acesso antecipado ao pacote. Tais características em conjunto são difíceis de ser encontradas em outros sistemas. eBPF também é conhecido por suas limitações devido às restrições da VM eBPF, responsáveis por garantir a integridade do sistema. Nesta seção são descritos os desafios e as limitações da tecnologia eBPF com base na análise realizada por [Miano et al. 2018].

Número de instruções limitado: 4096 é o número máximo de instruções que um programa eBPF pode ter para garantir que o programa termine de ser executado dentro do *kernel*. Essa restrição pode ser um problema quando funções de redes complexas são implementadas, pois o número de instruções geradas após o código C ser compilado pode ser maior que 4096.

Uma maneira de contornar essa limitação consiste dividir um programa com número de instruções maior que 4096 em vários subprogramas, e saltar de um subprograma para outro usando chamadas de calda (*tail calls*). Essa técnica habilita o desenvolvimento de serviços de rede como uma coleção de módulos fracamente acoplados, onde cada módulo realiza uma função diferente (análise, classificação ou modificação de campos) sem *overhead* ao saltar de um módulo para outro. Oito chamadas são o número máximo de chamadas subsequentes que não afetam o desempenho do eBPF.

Laço infinito: Programas eBPF antes de serem carregados no *kernel* são checados pelo verificador para garantir que programas não possam prejudicar o sistema. O verificador procura múltiplas ameaças (laço infinito, salto para trás, acesso a endereços inválidos, etc.). Quando uma ameaça é detectada, o programa automaticamente é rejeitado. Laços são proibidos em programas eBPF, pois um programa pode não terminar de ser executado dentro do *kernel*.

eBPF contorna essa limitação explorando o uso de diretivas `pragma unroll` do compilador LLVM, reescrevendo um laço como uma sequência repetida de instruções independentes. Essa técnica soluciona o problema. No entanto, ela aumenta o número de instruções, e para alguns casos pode não funcionar por não conseguir definir o valor da constante superior do término do laço, por exemplo, número máximo de cabeçalhos IPv6, rótulo MPLS alinhado e tamanho do pacote. Listamos três exemplos de aplicações no qual o uso do laço pode ser um problema:

1. **Análise de cabeçalhos alinhados:** No protocolo IPv6 percorrer todos os cabeça-

lhos de extensão é necessário para encontrar o último cabeçalho que indica o tipo do protocolo da camada superior na carga do pacote. Esse mesmo problema ocorre nos cabeçalhos MPLS e VLAN, no qual o número de instâncias não é conhecido a priori. Desenvolver funções de rede que realizam essas ações no eBPF não são possíveis de serem implementados sem introduzir restrições adicionais.

2. **Processamento da carga do pacote:** eBPF não permite escanear todo o pacote. No entanto, essa funcionalidade é requerida, por exemplo, para verificar a presença de uma assinatura na carga do pacote. Existem casos que percorrer a carga do pacote pode ser evitado devido ao uso de cabeçalhos específicos disponíveis no pacote que executam todo o trabalho, por exemplo, recalcular o *checksum* L3/L4.

3. **Busca linear estruturas de dados:** Algoritmos utilizam busca linear para encontrar determinado conteúdo em estruturas de dados. No eBPF, mapas são exemplos de estruturas de dados. Essas estruturas precisam ser adaptadas para permitir busca linear de um conteúdo. *Firewall* é um exemplo de aplicação que utiliza busca linear em um mapa para encontrar a regra que corresponde ao pacote. Neste exemplo, todas as políticas ativas são escaneadas através da busca linear.

Enviar o mesmo pacote para múltiplas portas: Requisições ARP, *multicast* e inundação são exemplos de funções de rede que precisam enviar o mesmo pacote para várias portas simultaneamente. No entanto, três problemas podem ser encontrados quando essa operação é implementada no eBPF. Primeiro, todas as interfaces precisam ser percorridas para encaminhar o pacote de acordo com número de vezes desejado. Essa operação pode apenas ser implementada se for possível definir o valor da constante do término do laço. Segundo, o pacote precisa ser clonado antes de ser enviado por uma interface adicional. Essa operação pode ser realizada utilizando a função auxiliar `bpf_skb_clone_redirect()`, duplicando simultaneamente e encaminhando o pacote original para interface adicional. No entanto, não existe uma função similar para programas XDP. Essa função está disponível apenas quando o programa eBPF está anexado ao gancho TC. Terceiro, se o serviço faz parte de uma cadeia virtual (*virtual chain*) composta por várias funções de rede virtuais conectadas por várias chamadas subsequentes, tal abordagem falha. Isso ocorre porque a função de redirecionamento segue por uma chamada subsequente, e nunca retorna o controle para o chamador, bloqueando o código do chamador de enviar pacote para várias portas.

Processamento de pacote dirigido a evento: A execução de um programa eBPF no *kernel* é acionada por evento. O único evento suportado por programas TC/XDP ocorre quando o quadro transita sobre um gancho selecionado dentro do *kernel*. Essa característica previne o plano de dados eBPF de reagir a outros eventos, tal como, tempo expirado de um pacote que sinaliza a necessidade de enviar outro pacote periodicamente, ou atualizar uma entrada que expirou na tabela. Tais eventos devem ser processados em outro lugar, por exemplo, no plano de controle ou no módulo caminho lento (*slow path*).

O módulo caminho lento é um novo componente que pode ser introduzido no plano de controle para executar código arbitrário, e lida com situações nas quais a versão atual do eBPF não pode ser utilizada. A funcionalidade deste módulo será receber pacotes

enviados do plano de dados eBPF, reagir de acordo com processamento arbitrário definido pelo desenvolvedor e encaminhar o pacote modificado para porta de saída.

Não suportar plano de controle complexo: eBPF apresenta um plano de controle simples composto por um conjunto de abstrações que ajudam desenvolvedores a criar código no plano de dados (ganchos, mapas, etc). Como consequência disso, *frameworks* eBPF existentes são primitivos em relação à tarefa de controle. Serviços de rede são diferentes de operações no espaço do usuário e mapas, requerendo um plano de controle mais sofisticado, que suporte o módulo caminho lento ou processe pacotes especiais (por exemplo, protocolos de roteamento). A simplicidade do plano de controle eBPF obriga desenvolvedores a dedicar horas no desenvolvimento de códigos de serviços rede por não terem um *framework* eBPF que auxilia no desenvolvimento de aplicações.

3.11. Conclusão

Neste trabalho apresentamos uma visão dos aspectos teóricos e práticos no desenvolvimento de pesquisa relacionado ao processamento rápido de pacotes. Na parte teórica, apresentamos as máquinas BPF e eBPF, a visão geral do sistema, ganchos e resultados de pesquisa recente. Na parte prática, focamos no eBPF e no gancho XDP. Mostramos como utilizá-los, exemplos e ferramentas. Dado os seus potencias no processamento rápido de pacotes, consideramos que existe um grande potencial no desenvolvimento de novos projetos de pesquisa com eBPF e XDP, seja como ferramenta para o desenvolvimento de novas funções de rede, seja permitindo prover novas funcionalidades no plano de dados, seja como ferramenta para o desenvolvimento de novos padrões e protocolos de comunicação, seja no desenvolvimento de novos protótipos de pesquisa, seja como alvo de estudos sobre novas soluções de rede. Certamente, eBPF e XDP ajudarão no desenvolvimento de novos trabalhos interessantes e com grande potencial na área de redes de computadores. Com certeza, mais trabalhos interessantes na área estão por vir.

Referências

[Abdelsalam et al. 2017] Abdelsalam, A., Clad, F., Filsfils, C., Salsano, S., Siracusano, G., and Veltri, L. (2017). Implementation of virtual network function chaining through segment routing in a linux-based NFV infrastructure. In *2017 IEEE Conference on Network Softwarization: Softwarization Sustaining a Hyper-Connected World: en Route to 5G, NetSoft 2017*, pages 1–5. IEEE.

[Ahmed et al. 2016] Ahmed, Z., Alizai, M. H., and Syed, A. A. (2016). Inkev: Inkernel distributed network virtualization for dcn. *ACM SIGCOMM Computer Communication Review*, 46(3).

[Baidya et al. 2018] Baidya, S., Chen, Y., and Levorato, M. (2018). ebpf-based content and computation-aware communication for real-time edge computing. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WORKSHOPS)*, pages 865–870.

[BCC 2019a] BCC (2019a). BPF Compiler Collection. `https://github.com/iovisor/bcc`.

[BCC 2019b] BCC (2019b). BPF program types. https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md\#program-types.

[BCC 2019c] BCC (2019c). Compiling P4 to EBPF. https://github.com/iovisor/bcc/tree/master/src/cc/frontends/p4.

[BCC 2019d] BCC (2019d). XDP compatible drivers. https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md#xdp.

[Beckett et al. 2018] Beckett, D., Joubert, J., and Horman, S. (2018). Host dataplane acceleration (hda).

[Bertin 2017] Bertin, G. (2017). Xdp in practice: integrating xdp into our ddos mitigation pipeline. In *Technical Conference on Linux Networking, Netdev*, volume 2.

[Bertrone et al. 2018] Bertrone, M., Miano, S., Risso, F., and Tumolo, M. (2018). Accelerating Linux Security with eBPF Iptables. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, SIGCOMM '18, pages 108–110, New York, NY, USA. ACM.

[Bloom 1970] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

[Bosshart et al. 2014] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

[bpftool 2018] bpftool, A. (2018). Manual bpftool. https://elixir.bootlin.com/linux/v4.18-rc1/source/tools/bpf/bpftool/Documentation/bpftool.rst.

[bpftrace 2019] bpftrace (2019). High-level tracing language for Linux eBPF. https://github.com/iovisor/bpftrace.

[Budiu 2015] Budiu, M. (2015). Compiling p4 to ebpf.

[Castanho et al. 2019] Castanho, M. S., Dominicini, C. K., and Vieira, M. A. M. (2019). Cadeia-Aberta: Arquitetura para SFC em Kernel usando eBPF. In *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, Porto Alegre, RS, Brasil. SBC.

[Cilium 2018] Cilium (2018). Cilium 1.0: Bringing the BPF Revolution to Kubernetes Networking and Security. https://cilium.io/blog/2018/04/24/cilium-10/.

[Cormode and Muthukrishnan 2005] Cormode, G. and Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75.

[Dargahi et al. 2017] Dargahi, T., Caponi, A., Ambrosin, M., Bianchi, G., and Conti, M. (2017). A survey on the security of stateful sdn data planes. *IEEE Communications Surveys & Tutorials*, 19(3):1701–1725.

[Duchene et al. 2018] Duchene, F., Jadin, M., and Bonaventure, O. (2018). Exploring various use cases for ipv6 segment routing. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, SIGCOMM '18, pages 129–131, New York, NY, USA. ACM.

[Dumazet 2011] Dumazet, E. (2011). A jit for packet filters. `https://lwn.net/Articles/437981/`.

[Facebook 2018] Facebook (2018). Katran source code repository. `https://github.com/facebookincubator/katran`.

[Feamster et al. 2014] Feamster, N., Rexford, J., and Zegura, E. (2014). The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98.

[Filsfils et al. 2018] Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and Shakir, R. (2018). Segment routing architecture. RFC 8402, RFC Editor.

[Flajolet and Martin 1985] Flajolet, P. and Martin, G. N. (1985). Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209.

[gobpf 2019] gobpf (2019). Go bindings for creating BPF programs. `https://github.com/iovisor/gobpf`.

[Google 2014] Google (2014). Speeding up and strengthening HTTPS connections for Chrome on Android. `https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html`.

[Halpern and Pignataro 2015] Halpern, J. and Pignataro, C. (2015). Service function chaining (SFC) architecture. RFC 7665, IETF.

[Høiland-Jørgensen et al. 2018] Høiland-Jørgensen, T., Brouer, J. D., Borkmann, D., Fastabend, J., Herbert, T., Ahern, D., and Miller, D. (2018). The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, pages 54–66, New York, NY, USA. ACM.

[IOvisor 2019] IOvisor (2019). Iovisor project. www.iovisor.org. Disponível em 29/03/2019.

[Jouet et al. 2015] Jouet, S., Cziva, R., and Pezaros, D. P. (2015). Arbitrary packet matching in openflow. In *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6.

[Jouet and Pezaros 2017a] Jouet, S. and Pezaros, D. P. (2017a). Bpfabric: Data plane programmability for software defined networks. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, ANCS '17, pages 38–48, Piscataway, NJ, USA. IEEE Press.

[Jouet and Pezaros 2017b] Jouet, S. and Pezaros, D. P. (2017b). BPFabric implementations. `https://github.com/UofG-netlab/BPFabric`.

[Krishnamurthy et al. 2003] Krishnamurthy, B., Sen, S., Zhang, Y., and Chen, Y. (2003). Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, IMC '03, pages 234–247, New York, NY, USA. ACM.

[libbpf 2018] libbpf, A. (2018). libbpf source code. `https://elixir.bootlin.com/linux/v4.18-rc1/source/tools/lib/bpf`.

[Maguire 2019] Maguire, A. (2019). Notes on bpf (1) - a tour of progam types. https://blogs.oracle.com/linux/notes-on-bpf-1.

[McCanne and Jacobson 1993] McCanne, S. and Jacobson, V. (1993). The bsd packet filter: A new architecture for user-level packet capture. In *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*, USENIX'93, pages 2–2, Berkeley, CA, USA. USENIX Association.

[McKeown et al. 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

[Miano et al. 2018] Miano, S., Bertrone, M., Risso, F., Tumolo, M., Bernal, M. V., and Tumolo, M. (2018). Creating Complex Network Services with eBPF: Experience and Lessons Learned. *High Performance Switching and Routing (HPSR). IEEE*, pages 1–8.

[Mijumbi et al. 2016] Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., De Turck, F., and Boutaba, R. (2016). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys and Tutorials*, 18(1):236–262.

[Monnet 2019] Monnet, Q. (2019). Rust virtual machine and JIT compiler for eBPF programs. `https://github.com/qmonnet/rbpf`.

[Netronome 2019] Netronome (2019). Sample bpf offload apps. `https://github.com/Netronome/bpf-samples`.

[Nir and Langley 2018] Nir, Y. and Langley, A. (2018). ChaCha20 and Poly1305 for IETF Protocols. RFC 8439.

[Pacífico et al. 2018] Pacífico, R. D. G., Coelho, G. R., Vieira, M. A. M., and Nacif, J. A. M. (2018). Roteador SDN em hardware independente de protocolo com análise, casamento e ações dinâmicas. In *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, Porto Alegre, RS, Brasil. SBC.

[ply 2019] ply (2019). Dynamic Tracing in Linux. `https://github.com/iovisor/ply`.

[Quinn et al. 2018] Quinn, P., Elzur, U., and Pignataro, C. (2018). Network service header (NSH). RFC 8300, IETF.

[Santos et al. 2019a] Santos, E. R. S., Câmara Júnior, E. P. M., and Vieira (2019a). Bpfabric implementing sketches. `https://github.com/elerson/BPFabric`.

[Santos et al. 2019b] Santos, E. R. S., Câmara Júnior, E. P. M., Vieira, M. A. M., and Vieira, L. F. M. (2019b). Aplicações de monitoramento de tráfego utilizando redes programáveis eBPF. In *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, Porto Alegre, RS, Brasil. SBC.

[Schulist et al. 2019] Schulist, J., Borkmann, D., and Starovoitov, A. (2019). Linux socket filtering aka berkeley packet filter (bpf). www.kernel.org/doc/Documentation/networking/filter.txt. Disponível em 17/03/2019.

[Sharma et al. 2017] Sharma, N. K., Kaufmann, A., Anderson, T. E., Krishnamurthy, A., Nelson, J., and Peter, S. (2017). Evaluating the power of flexible packet processing for network resource allocation. In *NSDI*, pages 67–82.

[Song 2013] Song, H. (2013). Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 127–132. ACM.

[Tu et al. 2017] Tu, C., Stringer, J., and Pettit, J. (2017). Building an extensible open vswitch datapath. *SIGOPS Oper. Syst. Rev.*, 51(1):72–77.

[ubpf 2019] ubpf (2019). Userspace eBPF VM. `https://github.com/iovisor/ubpf`.

[Vieira et al. 2019] Vieira, M. A. M., Pacífico, R. D. G., Castanho, M. S., Santos, E. R. S., Câmara Júnior, E. P. M., and Vieira, L. F. M. (2019). Tutorial eBPF. `https://github.com/mscastanho/bpf-tutorial`.

[VMWare 2018] VMWare (2018). p4c-xdp. `https://github.com/vmware/p4c-xdp`.

[Wang et al. 2017] Wang, H., Soulé, R., Dang, H. T., Lee, K. S., Shrivastav, V., Foster, N., and Weatherspoon, H. (2017). P4fpga: A rapid prototyping framework for p4. In *Proceedings of the Symposium on SDN Research*, pages 122–135. ACM.

[Yu et al. 2013] Yu, M., Jose, L., and Miao, R. (2013). Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42.

[Zilberman et al. 2015] Zilberman, N., Watts, P. M., Rotsos, C., and Moore, A. W. (2015). Reconfigurable network systems and software-defined networking. *Proceedings of the IEEE*, 103(7):1102–1124.

Capítulo 4

Análise de Dados em Redes Sem Fio de Grande Porte: Processamento em Fluxo em Tempo Real, Tendências e Desafios

Dianne S. V. Medeiros (UFF), Helio N. C. Neto (UFF),
Martin Andreoni Lopez (Samsung R&D), Luiz Claudio S. Magalhães (UFF),
Edelberto F. Silva (UFJF), Alex B. Vieira (UFJF),
Natalia C. Fernandes (UFF), Diogo M. F. Mattos (UFF)

Abstract

In this chapter, we focus on knowledge extraction from large wireless networks through stream processing. We present the primary methods of sampling, data collection and monitoring of wireless networks and we characterize knowledge extraction as a machine learning problem on big data stream processing. The Apache Spark and Apache Flink are the main trends on big data stream processing frameworks and, thus, are discussed in this chapter. We explore the data preprocessing, the feature engineering and the machine learning algorithms applied to the scenario of wireless network analytics. We address challenges and research projects in wireless network monitoring and stream processing. Finally, future perspectives, such as deep learning and reinforcement learning in stream processing, are anticipated.

Resumo

Este capítulo foca na extração de conhecimento em aplicações de redes sem fio de grande porte através do processamento de dados em fluxo. O capítulo apresenta os principais métodos de amostragem, coleta de dados e monitoramento de redes sem fio e caracteriza a extração do conhecimento como um problema de aprendizado de máquina em processamento de grandes massas de dados em fluxo. As plataformas Apache Spark e Apache Flink são as principais tendências entre plataformas para o processamento de dados em fluxo e, portanto, são analisadas neste capítulo. Discute-se o pré-processamento, a engenharia de características e os algoritmos para o aprendizado de máquina aplicados ao cenário de redes sem fio. Desafios e projetos de pesquisa em monitoramento de redes sem fio e processamento em fluxo são elencados. Por fim, perspectivas futuras, como a aprendizagem profunda e por reforço aplicadas a dados em fluxo, são antecipadas.

4.1. Introdução

A popularização de celulares inteligentes e dispositivos conectados à Internet das Coisas (*Internet of Things* - IoT) [Mattos et al., 2018] impulsionou o crescimento da geração de dados provenientes de dispositivos móveis em redes sem fio. Nos últimos 5 anos, as redes móveis alcançaram um crescimento acumulado de dezoito vezes, registrando 63% de aumento no volume de dados trafegado no ano de 2016 em relação ao ano anterior [Cisco, 2017]. O crescente volume de tráfego nas redes móveis também impacta redes sem fio fixas, já que a terceirização do tráfego de dados de dispositivos móveis para redes sem fio fixas compõe 60% do tráfego sem fio total. Assim, as redes IEEE 802.11 representam o principal meio de acesso de uma parcela significativa dos usuários finais, nos mais variados ambientes. Em universidades e em empresas, grande parte dos usuários conecta à Internet através da rede sem fio institucional ou a serviços internos através de intranet sem fio. A rede sem fio institucional da Universidade Federal Fluminense (UFF), por exemplo, conta com a operação de 547 pontos de acesso que atendem a uma população de mais de 60 mil usuários, com picos de 5 mil usuários conectados concomitantemente, gerando mais de 100 Mb/s de dados a serem analisados [Magalhães e Mattos, 2018]. Os dados de gerenciamento e monitoramento dessas redes contêm conhecimento sobre os usuários, as redes e os padrões de uso e de deslocamento [Divgi e Chlebus, 2013]. Assim, as redes sem fio tornam-se excelentes fontes geradoras de dados.

Alinhado a essa tendência, há o crescente interesse em medições do desempenho fim a fim e seu impacto em aplicações móveis [Goel et al., 2015]. O monitoramento das redes móveis oferece informações úteis para pesquisadores e traz benefícios aos operadores de redes. A ausência de mecanismos de inteligência e de ação rápida na rede, muitas vezes associada a visões limitadas de ferramentas de controle por fluxo, prejudicam a qualidade de experiência (QoE) de usuários de redes sem fio de grande escala e, em especial, a extração de conhecimento sobre os usuários e as redes [Jang et al., 2017].

A popularidade das redes sem fio fomenta, então, o desafio de processar e gerenciar o grande volume de dados gerado, já que um único ponto de acesso suporta, tipicamente, algumas dezenas de clientes conectados [Magalhães e Mattos, 2018]. O monitoramento de redes sem fio também apresenta diversos desafios quando comparado ao monitoramento de redes cabeadas, tornando a tarefa ainda mais complexa. A replicação dos métodos usados tradicionalmente, realizando medidas dos parâmetros após os dados passarem para a rede cabeada, não revela o estado real da rede sem fio. Os métodos atuais não permitem distinguir, por exemplo, uma rede ociosa de uma que está tão congestionada que nenhum quadro está sendo entregue. Propostas para obter o estado das redes consideram medidas ativas, mas implicam alterações nos parâmetros avaliados. Medidas indiretas, como medir o uso do canal através dos contadores dos próprios pontos de acesso ou usar sensores para análise espectral e pontos para captura e análise de quadros, são empregadas ao custo de informações menos precisas. Além disso, metadados coletados nas redes permitem a criação de aplicações que tornam o ambiente ciente de seu contexto, auxiliando no monitoramento em todos os níveis. É válido ressaltar que a configuração de uma rede sem fio de larga escala é uma atividade complexa, tanto pelo número de elementos a serem configurados, como pelos fatores de propagação rádio e interferências, que não são simples de serem estimados *a priori*. Assim, o monitoramento é uma ferramenta chave para entender a topologia *de facto* da rede. A captura dos *beacons*

recebidos através de varredura passiva, associada à potência com que foram recebidos, permite identificar quais pontos de acesso são vizinhos e a distância rádio entre eles. A coleta de dados permite ainda a execução de operações básicas em redes, como cobranças de serviços, detecção de ameaças, isolamento e mitigação de falhas.

As redes móveis sem fio fornecem também informação espaço-temporal sobre usuários e condições da rede para dotar o sistema de visibilidade e inteligência fim a fim, permitindo melhor compreensão da dinâmica da rede a longo prazo. Informações sobre geolocalização [Cici et al., 2015] ou sobre o posicionamento dos usuários [Alessandrini et al., 2017] permitem identificar padrões de uso e detectar anomalias. Além disso, a análise dos dados fornecidos por essas redes habilita a auto-coordenação das funções e entidades da rede e a construção de redes mais eficientes e proativas. A análise do grande volume de dados provenientes das redes sem fio é desafiadora devido às características inerentes à própria rede, como a mobilidade dos usuários, o ruído presente e a redundância dos dados coletados que impactam diretamente as cinco dimensões fundamentais do processamento de grande massas de dados, volume, velocidade, variedade, valor e veracidade. Nesse contexto, algumas técnicas tradicionais de processamento de grandes massas de dados podem ser usadas, mas é necessário empregar técnicas de processamento de dados em fluxo para análise da rede em tempo real.

O processamento de grandes massas de dados em fluxo consiste no tratamento de dados potencialmente não limitados em número de amostras, isto é, que chegam a todo momento, e não limitados quanto a espaço de atributos, ou seja, o universo de atributos dos dados assim como a distribuição estatística são desconhecidos. A ideia de processamento em fluxo contrapõe-se ao processamento em lotes, em que um conjunto de dados bem limitado e conhecido é processado de uma só vez por uma plataforma de processamento de grande massa de dados. O processamento em lote exige grande disponibilidade de memória para armazenamento dos dados e implica maior latência na geração de respostas de processamento. A principal característica do processamento de dados em fluxo frente ao processamento em lotes é a menor latência no tempo de processamento de cada amostra de dados. Contudo, o processamento em fluxo impõe restrições de memória para o armazenamento dos dados entrantes e requer que algoritmos tradicionais de aprendizado de máquina sejam adaptados ao cenário em que o conjunto de dados de treinamento não pode ser revistado devido ao número ilimitado de amostras e ao requisito de latência mínima ao processar cada nova amostra. Para tanto, plataformas de processamento de dados em fluxo, como Apache Spark Streaming [Zaharia et al., 2013] e Apache Flink [Carbone et al., 2015b], propõem dois modelos distintos de processamento de amostras em fluxo, o processamento em micro-lotes e o processamento por amostra.

Ao adotar mecanismos de aprendizado de máquina para extrair conhecimento de dados em fluxo, os algoritmos estão sujeitos a erros na aprendizagem em reação a mudanças de conceito nos dados de entrada [Andreoni Lopez et al., 2019]. Nesse sentido, aplicações de aprendizado de máquina devem ser conscientes da distribuição estatística dos atributos dos dados de entrada e devem verificar quando há uma mudança nas distribuições. As aplicações de aprendizado de máquina para processamento de dados em fluxo podem considerar que os dados entrantes são rotulados por algum processo que gere uma verdade básica, ou que existe um modelo de aprendizado previamente definido ou, ainda, podem buscar padrões ocultos nos dados sem que nenhum modelo seja fornecido.

O objetivo deste capítulo é apresentar os principais algoritmos e técnicas de processamento em fluxo, *stream processing*, em tempo real para extração de conhecimento do monitoramento de redes sem fio em grande escala, chamado de *Wi-Fi Analytics*. É apresentada uma visão teórica do processamento de dados em fluxo e do uso de algoritmos de aprendizado de máquina com treinamento em tempo real. Além disso, são discutidas algumas aplicações dos algoritmos no monitoramento de redes sem fio. O monitoramento das redes sem fio consiste tanto na caracterização do tráfego gerado pelos usuários do serviço quanto na avaliação espectral em tempo real. Tais vertentes de monitoramento geram grandes massas de dados, que devem ser processadas em tempo real, e o resultado do processamento permite identificar padrões de uso, definir perfis de usuários, identificar falhas ou queda de desempenho em pontos específicos da rede ou otimizar alocações de canais. Nesse sentido, o monitoramento e o controle de redes sem fio de grande porte compõem um cenário que exige o processamento em tempo real e de respostas imediatas para a adequação da rede a picos de uso e concentrações esporádicas de usuários. Assim, no cenário de análise de dados em redes sem fio de grande porte é essencial o uso dos algoritmos de processamento em fluxo.

Este capítulo está organizado da seguinte forma. A Seção 4.2 apresenta ferramentas e métodos para a gerência e o monitoramento de redes sem fio. A Seção 4.3 define as etapas para a realização da extração de conhecimento em grandes massas de dados. As plataformas de processamento de dados em fluxo Apache Spark e Apache Flink são exploradas na Seção 4.4. O aprendizado de máquina em fluxo e algoritmos de aprendizado incremental são detalhados na Seção 4.5. A Seção 4.6 elenca os desafios de pesquisa e as perspectivas futuras na análise de grandes massas de dados gerados por redes sem fio. Por fim, a Seção 4.7 conclui o capítulo.

4.2. Monitoramento de Redes Sem Fio

Grandes quantidades de dados geradas a partir de dispositivos móveis e que não podem ser processadas em uma única máquina constituem o conceito de *Mobile Big Data* (MBD) [Guo et al., 2018]. A Figura 4.1 mostra um esquema das camadas que compõem o MBD. Na primeira, Fontes, ocorre a geração de dados a partir da coleta por parte de dispositivos móveis utilizando tecnologias como Wi-Fi, GPS e comunicação celular. Na segunda camada, Integração, os dados das diferentes fontes heterogêneas são integrados. Primeiramente, os dados são validados e em seguida são unificados. Para esse fim, os dados são pré-processados, usando diferentes técnicas aplicadas para a limpeza e integração. Na terceira camada, Preparação, o pré-processamento dos dados continua, preparando-os para o consumo na camada de analíticas. Na quarta camada, Analíticas, a análise dos dados é feita através de diferentes algoritmos para identificação de padrões, classificação, entre outros. Finalmente, na quinta camada, Aplicação, os resultados produzidos pela quarta camada são usados por diferentes aplicações.

Trabalhos no contexto de MBD focam na localização de objetos combinando o uso de aprendizado de máquina para redes sem fio. Essas técnicas compõem a Ciência de Análise do Wi-Fi (*Wi-Fi Analytics*), e utilizam pacotes de gerenciamento, como pedido/resposta de *probes* e *beacons* do protocolo IEEE 802.11 para determinar a localização e deslocamento de objetos [Acer et al., 2015, Acer et al., 2016]. Os algoritmos utilizados para estimar localização são as Máquinas de Vetores de Suporte (*Support Vector*

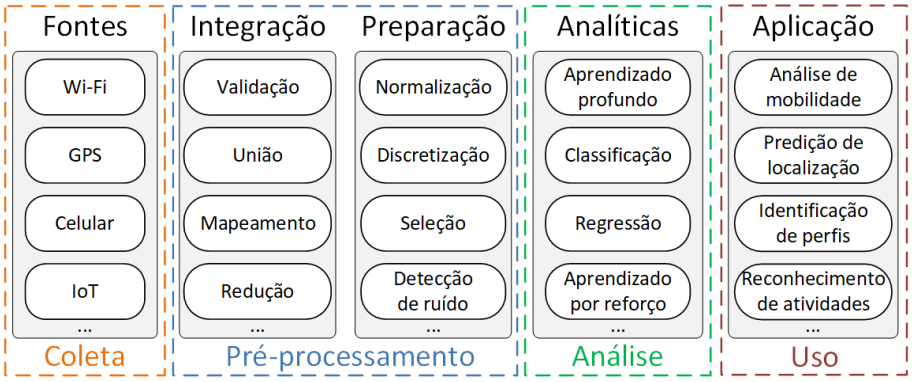


Figura 4.1. Esquema do processo de *Mobile Big Data*. A primeira camada gera os dados, enquanto a segunda integra os dados provenientes de diferentes fontes. Na terceira e quarta camadas esses dados são pré-processados para serem consumidos pela quinta camada, na qual diferentes algoritmos são aplicados para analisar os dados. Por fim, diferentes aplicações fazem uso dos resultados das análises realizadas.

Machine - SVM), com *kernel* gaussiano. As informações físicas, como intensidade do sinal contidas nos *probes*, são obtidas dos *gateways* Wi-Fi, podendo estimar, por exemplo, a movimentação de multidões. A mobilidade deduzida a partir de grandes massas de dados em geral combina diferentes sinais sem fio, como serviços de posicionamento global (GPS), sinais celulares como GSM e LTE, sinais Wi-Fi, entre outros, para determinar a geolocalização de pessoas e as trajetórias seguidas de forma acurada [Xu et al., 2017]. Nesse contexto, é possível também determinar a localização e as trajetórias de veículos, de forma que sistemas de tráfego inteligente que utilizam ferramentas de *Big Data* podem analisar dados provenientes de diferentes fontes sem fio para mitigar problemas de engarrafamento [Chen et al., 2018]. Não havendo dados provenientes de outras redes sem fio além da rede Wi-Fi, a acuidade da localização de objetos e pessoas dependerá da densidade de pontos de acesso existentes na rede. Quanto menos densa a rede, menor é a acuidade da localização encontrada. Os dados de mobilidade coletados são analisados por aprendizagem profunda, classificação e regressão para adquirir conhecimento útil como a previsão do fluxo de tráfego de veículos. Os algoritmos utilizados são os modelos de séries temporais, previsão baseada em aprendizado profundo, modelos de cadeias de Markov e combinação de redes neurais (*Neural Networks* - NNs) e média móvel integrada autorregressiva (*Autoregressive Integrated Moving Average* - ARIMA).

A ciência de análise do Wi-Fi também pode ser usada para melhorar a qualidade de experiência (*Quality of Experience* - QoE) dos usuários. Moura *et al.* apresentam uma combinação de aprendizado de máquina com as redes definidas por software (*Software Defined Networking* - SDN) para monitorar e controlar o congestionamento dos canais das redes locais sem fio (*Wireless Local Area Networks* - WLANs) [Moura et al., 2019]. A técnica de aprendizado de máquina utilizada é o bandido de k-braços (*Multi-Armed Bandit*), que é um algoritmo de aprendizado por reforço. O algoritmo funciona como realimentação para o controlador SDN para melhorar a experiencia do usuário no acesso de páginas *Web*. A ciência de análise do Wi-Fi no contexto de MDB permite, então, caracterizar as redes sem fio IEEE 802.11, através da identificação dos atores e fatores presentes no ambiente a partir de dados coletados. Essa caracterização pode estabelecer desde o posicionamento de usuários em ambientes *indoor* e *outdoor*, até a identificação de

pontos de acesso e redes concorrentes do meio a partir da gerência do espectro. Diversas abordagens podem ser seguidas para esse fim, como a caracterização da topologia da rede através do monitoramento de pontos de acesso e a caracterização do espectro. Em todo caso, é necessário utilizar ferramentas específicas para coletar os dados.

4.2.1. Caracterização de redes sem fio

O ambiente no qual as redes sem fio estão imersas está em constante mudança. Apesar dos pontos de acesso que formam a infraestrutura da rede normalmente estarem fixos e conectados à estrutura da rede cabeada, a mobilidade dos usuários causa flutuação no nível de ruído ao qual os pontos de acesso estão submetidos e interferem no ambiente rádio, já que os seres humanos são uma barreira para a propagação de micro-ondas. Em redes de grande porte, novos pontos de acesso podem ser ligados e desligados a qualquer tempo, alguns pontos de acesso podem ser dotados de mobilidade devido à proliferação do compartilhamento da Internet celular via Wi-Fi e o ambiente rádio pode ser modificado ainda pela abertura e fechamento de portas ou outras alterações do meio físico. Dessa forma, a caracterização da rede se torna uma tarefa complexa. Uma primeira abordagem para caracterizar a rede é realizar um estudo de vizinhança. Uma das formas de identificar a vizinhança de um ponto de acesso é coletar os *beacons* recebidos [Magalhães e Mattos, 2018]. Todo ponto de acesso envia quadros *beacon* periodicamente. O *beacon* faz parte da sincronia da rede sem fio, e de vários mecanismos, como os de economia de energia. Por isso nenhum ponto de acesso pode suprimir o seu envio, apesar de ser possível não incluir o nome da rede nesses quadros. Capturar os *beacons* permite, então, a descoberta de todos os pontos de acesso em uma área. Essa é uma forma de varredura passiva, mas pode ser estimulada pelo envio de quadros de `probe-request`, que gera a criação e envio de quadros `probe-response`. Seja a varredura passiva ou estimulada, esse tipo de coleta de dados identifica apenas os pontos de acesso.

A medida da potência recebida em um *beacon* é um indicador da distância rádio do emissor ao receptor. Como o ambiente varia continuamente, uma única medida não é confiável, mas com várias medidas é possível criar um mapa confiável das distâncias rádio entre pontos de acesso. Em uma rede sem fio institucional essas medidas podem ser usadas para criar uma configuração dos canais e da potência para cada ponto de acesso a fim de minimizar a interferência entre pontos de acesso vizinhos, sejam eles pertencentes à rede ou gerenciados por terceiros [Magalhães e Mattos, 2018]. Os pontos de acesso pertencentes à rede institucional têm conhecimento das estações associadas a eles e podem trocar essas informações com um sistema de monitoramento centralizado. Dessa forma, é possível para um operador saber o número de usuários associados a cada ponto de acesso e o número global de usuários associados à rede. No entanto, uma rede sem fio geralmente não tem conhecimento das estações associadas a outra rede sem fio.Para saber quantos usuários estão associados a redes vizinhas é necessário capturar pacotes para identificar os MACs de origem. Para inferir interferências que não sejam causadas por redes IEEE 802.11, tal como as causadas por dispositivos bluetooth, telefones sem fio ou fornos de micro-ondas, são necessários mecanismos como a varredura espectral.

O conhecimento da topologia da rede além de permitir explorar características relacionadas ao ambiente rádio, permite obter informações sobre os usuários. A localização de um usuário pode ser feita em baixa resolução assumindo que ele se encontra na vizi-

nhança do ponto de acesso ao qual está associado. De forma geral, como uma estação fica associada a apenas um ponto de acesso a cada instante, é possível usar o histórico ou sequência de associações a pontos de acesso para melhorar a acurácia da localização. No entanto, devido aos algoritmos usados atualmente para a escolha de pontos de acesso causarem muitas trocas mesmo quando o usuário está estático [Balbi et al., 2012], devido às flutuações da potência de sinal recebida provocadas pela variabilidade do ambiente de rádio, o histórico de associações apresenta precisão limitada. Assim, é necessário monitorar os pontos de acesso para descobrir a topologia da rede, caracterizar o ambiente rádio e os usuários associados à rede.

Monitoramento de pontos de acesso

Redes sem fio de grande porte necessitam de um controle centralizado para tomada de decisões, como a utilização do espectro e a alocação de clientes. O meio sem fio pode sofrer interferência de diversas fontes e, nesse ponto, a alocação de canais é um desafio. A complexidade do problema cresce ao se adicionar a restrição de prover um desempenho aceitável para usuários na sobreposição de canais [Luiz et al., 2015]. Canais próximos geram interferências e diminuem o desempenho das redes sem fio em suas vizinhanças. Portanto, é necessário que o projeto de instalação de redes sem fio considere a alocação de canais tanto do ponto de acesso instalado, como dos demais pontos de acesso já presentes na área. A alocação de canais ótima é um problema NP-Difícil [Leung e Kim, 2003], mas heurísticas alcançam boas soluções de configuração de canais [Maturi et al., 2017].

Em redes sem fio de grande porte, alternativas proprietárias para o gerenciamento de pontos de acesso, e consequente atribuição de canais, têm bom desempenho com a contrapartida de um alto custo financeiro [Balbi et al., 2012]. Para evitar esse elevado custo, *Balbi et al.* propõem um algoritmo para alocação de canais que considera primeiro a interferência interna, entre os pontos de acesso controlados, e depois a interferência externa, dos pontos de acesso com todos os pontos de acesso nas vizinhanças. Maturi *et al.* propõem um esquema de seleção dinâmica de canais que permite que uma rede IEEE 802.11 salte sobre os canais disponíveis sempre escolhendo o que tem menor utilização [Maturi et al., 2017]. Os autores implementam a proposta e mostram a viabilidade de realizar o salto em frequências para redes IEEE 802.11 com equipamento padrão de mercado. Outras propostas realizam a alocação de canal através da interferência observada entre pontos de acesso [Lin et al., 2017, Monteiro et al., 2016] e entre clientes e pontos de acesso [Luiz et al., 2015]. É importante também considerar o impacto do *handoff* nas redes sem fio de grande porte na qualidade da rede percebida pelo usuário. Nesse contexto, Shin *et al.* descrevem a rede sem fio através de um grafo e investigam como diminuir a latência durante o *handoff* [Shin et al., 2004]. A ideia central é desenvolver algoritmos que permitam que o *handoff* ocorra sem que a estação tenha a necessidade de ficar monitorando todos os canais. Para tanto, a estação armazena o conjunto de canais que cada vizinho está operando e o conjunto de pontos de acesso vizinhos em cada canal.

Huang *et al.* utilizam plataformas de MBD, ferramentas de análise de dados e aquisição distribuída para monitoramento de pontos de acesso sem fio na rede 3G WCDMA da Unicom, na China [Huang et al., 2014]. Os autores utilizam uma plataforma de armazenamento e análise de dados baseada no Sistema de Arquivos Distribuídos Hadoop (*Hadoop Distributed File System* - HDFS). Uma arquitetura semelhante à utilizada

pelos autores pode ser empregada para processar os dados de outras redes sem fio de grande porte, como redes Wi-Fi institucionais. Hadi *at al.* apresentam diversos trabalhos relacionados ao tema [Hadi et al., 2018].

Caracterização do ambiente rádio

As redes sem fio de próxima geração, como o 5G, devem suportar taxas de transmissão de dados extremamente altas e paradigmas de aplicações diversos e novos, que exigirão novas tecnologias rádio sem fio. O desafio é ajudar a rede sem fio no aprendizado adaptativo e inteligente na tomada de decisões, de modo que os diversos requisitos dessas redes possam ser atendidos. O aprendizado de máquina é uma das ferramentas de inteligência artificial mais promissoras, concebidas para suportar terminais rádio inteligentes. Espera-se que futuros terminais móveis inteligentes habilitados para operar em 5G acessem autonomamente as melhores bandas espectrais com o auxílio de aprendizagem e realizem inferência da eficiência espectral de forma sofisticada, controlando desde a potência de transmissão, ao ajuste da eficiência energética e protocolos de transmissão com o auxílio de aprendizagem e inferência de qualidade de serviço. Nesse sentido, Jiang *et al.* classificam diversos tópicos e trabalhos relacionados ao aprendizado de máquina aplicado à caracterização e análise de espectro em redes sem fio da próxima geração [Jiang et al., 2017]. O trabalho classifica as técnicas em aprendizado supervisionado, não-supervisionado e por reforço. Destaca-se que no contexto de caracterização de espectro diversas técnicas distintas podem ser aplicadas, tendo destaque os algoritmos com aprendizado supervisionado baseado em redes Bayesianas, modelos de regressão, KNN (*K-Nearest Neighbor*) e SVM (*Support Vector Machines*).

A captura de quadros sofre com as limitações da necessidade de uma boa recepção e de sintonização com o canal para captura com sucesso, apesar de haver uma probabilidade pequena de captura de quadros em canais adjacentes. Assim, transmissões que podem interferir com a rede sem fio, como *bluetooth*, fornos de micro-ondas, telefones sem fio, babás eletrônicas e mesmo quadros que colidiram ou que estejam abaixo do limite de sensibilidade dada a relação sinal-ruído, podem impedir que os quadros sejam vistos pela captura. Assim, uma alternativa à captura de quadros é o monitoramento espectral, que captura toda a energia que chega na antena na faixa de espectro sintonizada, independente da fonte. A captura do espectro é feita através de analisadores de espectro, como WiSpy[1]. Essa captura provê uma melhor visão das fontes de interferência no ponto monitorado e permite definir se uma fonte é um transmissor de uma rede sem fio ou de uma rede *bluetooth*, por exemplo. Isso é possível graças às "assinaturas" de cada tecnologia, isto é, o formato do espectro capturado. Uma limitação da captura espectral é não permitir resposta instantânea, uma vez que para identificar a "assinatura" é necessário uma integração no tempo. Além disso, existe um elevado custo financeiro associado à utilização de analisadores de espectro, principalmente em redes de grande porte.

Os analisadores de espectro de baixo custo como o WiSpy são constituídos basicamente por um *chipset* que implementa o IEEE 802.11 com um *firmware* específico capaz de usar o processador de sinais digitais do Wi-Fi na captura de energia em uma pequena faixa do espectro. Seguindo essa mesma ideia, é possível acessar o analisador de espectro no *chipset* Atheros de pontos de acesso que usam a distribuição *OpenWRT*, permitindo

[1]Disponível em https://www.metageek.com/products/wi-spy/

transformar pontos de acesso comuns em analisadores de espectro de baixo custo. O processo de captura utilizando esses pontos de acesso é feito em *hardware*, analisando cada quadro recebido. As medidas são extremamente rápidas e a quantidade de dados coletada é limitada, principalmente pelo tempo de transmissão e armazenamento.

Diversos trabalhos usam a caracterização do espectro em uma área como ferramenta para melhorar o desempenho das redes. Wang *et al.* propõem a otimização do uso do espectro das redes sem fio através da cooperação entre pontos de acesso para realizar formatação de feixe (*beamforming*) [Wang et al., 2018]. Os pontos de acesso são conectados via Ethernet para tornar mais ágil a troca de informações. A proposta baseia-se em três pilares: um esquema cooperativo que permite o sistema estimar desvios de fase em cada símbolo transmitido e dinamicamente ajustar as fases para garantir o alinhamento; um mecanismo de estimação que mede a qualidade do canal; e um algoritmo aleatório de escolha de usuários para realizar formatação de feixe com custo computacional constante. Os autores mostram que o algoritmo aleatório é capaz de escalar a rede linearmente e tem desempenho equivalente a 70% comparado a algoritmos mais complexos.

Caracterização do comportamento de usuários

Os padrões de mobilidade humana refletem muitos aspectos da vida, desde a disseminação global de doenças infecciosas até o planejamento urbano e padrões diários de deslocamento. Nos últimos anos, a prevalência de métodos e tecnologias de posicionamento, tais como o sistema de posicionamento global, geo-posicionamento de torre de rádio celular e sistemas de posicionamento Wi-Fi, têm direcionado esforços para coletar dados de mobilidade humana e extrair padrões de interesse dentro desses dados, com o objetivo de promover o desenvolvimento de serviços e aplicações baseados em localização. Os esforços para extrair padrões significativos em dados de mobilidade em grande escala têm solicitado o uso de técnicas de análise avançadas para mineração de dados, geralmente baseadas em métodos de aprendizado de máquina.

A abordagem para análise de posicionamento baseada em impressão digital (*fingerprints*) é comumente usada para posicionamento *indoor* e consiste de duas fases: uma *online* (treinamento) e outra *offline* (posicionamento). Na fase *offline*, um mapa rádio é criado usando valores de Força de Sinal Recebido (*Received Signal Strength* - RSS) que são medidos nos pontos de acesso existentes no ambiente. Os mapas de rádio incluem, além dos valores RSS, informações dos pontos de acesso em que as medidas foram feitas. Na fase *online* a localização é realizada combinando os valores RSS do mapa de rádio e os valores de RSS medidos pela unidade móvel. É interessante destacar que mudanças físicas de dispositivos entre as fases *offline* e *online* podem afetar a precisão do posicionamento, assim como a escolha de algoritmos e seus parâmetros na fase *online*. Na literatura de posicionamento, os algoritmos de aprendizado de máquina têm amplo uso na estimativa dessas posições. Diversos são os algoritmos sugeridos nessa aplicação, sendo uma tarefa não trivial a descoberta daquele que melhor se comporta em um dado cenário. Na comparação entre os algoritmos com relação ao posicionamento e tempo de computação, Bokzurt *et al.* mostram que o algoritmo k-vizinhos mais próximos (*K-Nearest Neighbors* - k-NN) é o mais adequado [Bozkurt et al., 2015] quando comparado aos algoritmos de Árvore de Decisão, Naïve Bayes, Rede Bayesiana, Otimização Sequencial Mínima (*Sequential Minimal Optimization* - SMO), AdaBoost e Bagging.

A disponibilidade de grandes conjuntos de dados utilizados para o rastreamento da localização do usuário se tornou mais realista a partir da chegada das diversas tecnologias relacionadas às telecomunicações e associadas à realidade do *Mobile Big Data* (MBD). Estudos analisam padrões de mobilidade usando modelos estatísticos que capturam propriedades gerais desses padrões observados. Gonzalez *et al.* analisa 100.000 traços (*traces*) de telefones celulares e identifica que a distância entre os usuários segue uma distribuição de probabilidade de lei de potência. Após a identificação dos padrões principais de movimentação de pessoas e do grau de previsibilidade [Song et al., 2010], abriu-se caminho para a pesquisa e aplicação da análise da mobilidade. Outra área interessante se baseia em teoria da complexidade de redes e ferramentas estatísticas. Esses estudos têm como foco a análise das relações sociais a fim de agregar padrões de mobilidade em grandes massas de dados.

Toch *et al.* classificam aplicações de caracterização de mobilidade de usuários em três categorias: (1) aplicações de modelagem do usuário; (2) aplicações de modelagem da localidade; (3) aplicações de modelagem da trajetória [Toch et al., 2018]. Em (1) é analisado apenas um único usuário por um período de tempo, objetivando prever seu padrão de mobilidade e sua localização futura. Em (2) apenas localidades, ou áreas, são levadas em consideração, visando prever a quantidade de pessoas que devem passar por uma certa área. Já em (3) uma análise espaço-temporal de pontos é criada para identificar padrões de mobilidade do usuário, tentando identificar agrupamento de usuários com perfis semelhantes e sua trajetória futura.

É natural imaginar a MBD associada a essas aplicações, assim como a necessidade de um processamento eficiente em relação ao tempo de resposta. Dessa forma, a abordagem da modelagem é de suma importância. Diferentes abordagens usam diferentes métodos, o que influenciará diretamente no resultado. A maioria dos métodos se baseia em aprendizado de máquina, seja de forma supervisionada ou não. Outros métodos utilizam análise de séries temporais não lineares, modelos Markovianos ou regressão.

4.2.2. Principais ferramentas para coleta de dados em redes sem fio

Abordagens de monitoramento de rede são classificadas em ativas ou passivas. As abordagens ativas, implementadas por ferramentas comuns, como o `ping` e `traceroute`, injetam tráfego em uma rede para executar diferentes tipos de medições. As abordagens passivas, por outro lado, não interferem diretamente no tráfego da rede. As ferramentas que adotam tal abordagem apenas observam o tráfego existente à medida que ele passa por um ponto de medição. Esse método fornece mais informações sobre o tráfego de rede, uma vez que pacotes completos podem ser capturados e analisados [Hofstede et al., 2014]. Exportar apenas informações de fluxos é uma variação de abordagem passiva de monitoração para torná-la mais escalável. Nessa abordagem, pacotes de rede são agregados em fluxos e os dados sobre os fluxos são exportados para armazenamento e análise futura. Um fluxo é definido como um conjunto de pacotes passando por um ponto de observação na rede durante um certo período de tempo, de tal forma que todos os pacotes apresentem um conjunto de propriedades comuns [Hofstede et al., 2014]. Essas propriedades comuns podem incluir campos de cabeçalho de pacote, como origem e destino, endereços IP, números de portas, tipo de protocolo de transporte, entre outros. Neste capítulo, discute-se algumas das ferramentas mais importantes, como SNMP, Net-

Flow, sFlow e IPFIX, apresentando suas principais características, limitações e uso em redes sem fio.

O **SNMP** é um protocolo de gerência da camada de aplicações da pilha de protocolos TCP/IP, que utiliza serviços do protocolo de transporte UDP para enviar e receber suas mensagens através da rede. Resumidamente, o protocolo é utilizado para obter informações de servidores SNMP, através de requisições que um gerente faz a um agente. Os agentes são instalados em determinados dispositivos de rede, sobre os quais existe interesse de monitoramento. Cada elemento de rede tem variáveis que representam suas informações e seu estado. O gerente pode acessar essas informações e, até mesmo, mudar o estado de alguma propriedade do elemento que está sendo monitorado. Nesse sentido, cada elemento de rede gerenciado deve ter um agente SNMP e uma base de informações de gerenciamento (*Management Information Base* - MIB). A MIB contém informação sobre os objetos gerenciados, que são abstrações do mundo real representando recursos do sistema que poderão ser consultados ou alterados. Os objetos gerenciados podem ter permissões para consulta (leitura) e até mesmo de alteração. Nesse sentido, leituras representam o estado atual do recurso do elemento de rede monitorado e alterações refletem no recurso monitorado em si. Por exemplo, pode-se usar SNMP para acessar (leitura) configurações de um ponto de acesso, tais como interfaces ativas, canal em uso e potência.

Atualmente existem três versões principais do protocolo (SNMPv1, SNMPv2 e SNMPv3). O SNMP apresenta algumas limitações bem conhecidas. o protocolo, por exemplo, não é apropriado para o gerenciamento de redes muito grandes. Como ele é baseado em um mecanismo de sondagem (*polling*), gerenciar muitos elementos de rede pode provocar atraso excessivo. Ademais, o SNMP básico não tem mecanismos de autenticação. Por fim, destaca-se que o uso de SNMP gasta recursos computacionais, de forma que aplicá-lo em redes sem fio pode exaurir os recursos dos elementos de rede. Nesse sentido, SNMP não é apropriado para redes de sensores sem fio.

O **NetFlow** é um protocolo de rede proprietário da Cisco usado para análise de fluxos. Ele permite a coleta e agregação de informações sobre o tráfego de rede que entra ou sai de um elemento de rede que tem o protocolo habilitado.Os registros de informações coletados pelo NetFlow são enviados por mensagens do protocolo a um local centralizado na rede. Nesse local, os dados podem ser analisados por um administrador de rede que pode determinar, entre outros, a origem e o destino do tráfego, as classes de serviço de tráfego na rede e causas de possíveis congestionamento na rede. Nesse sentido, exemplos do uso típico das estatísticas coletadas pelo protocolo NetFlow são: (i) monitoramento de banda em enlances; (ii) detecção de ameaças em redes, como ataques de negação de serviço (*Denial of Service* - DoS); (iii) contabilidade de uso e cobrança; (iv) investigação das causas de congestionamento e lentidão dos recursos de rede e aplicações. A Figura 4.2 mostra uma configuração típica de monitoramento usando o NetFlow, que consiste em três componentes principais [Hofstede et al., 2014]. O Exportador NetFlow agrega pacotes em fluxos e exporta registros de fluxos para um ou mais coletores de fluxo. A agregação é feita com base nos endereços IP de origem/destino, portas de origem/destino, classes de serviço, protocolo IP e interface de origem. O Coletor NetFlow é responsável pela recepção, armazenamento e pré-processamento de dados de fluxos recebidos de um exportador de fluxos. Já o Analisador analisa os dados de fluxos recebidos. Essas análises cobrem contextos, como detecção de intrusão e geração de perfis de tráfego.

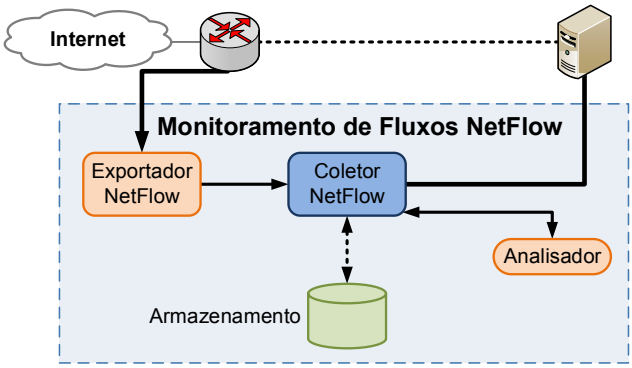


Figura 4.2. Configuração típica de monitoramento de fluxo usando o NetFlow. Os dispositivos habilitados com o protocolo NetFlow, exportadores NetFlow, criam registros, agregando os pacotes em fluxos e exportando os registros para coletores de fluxo. Os coletores NetFlow armazenam e pré-processam os dados recebidos e encaminham para o analisador que consome os dados para gerar informação.

Cada pacote que será encaminhado é examinado e o primeiro pacote que se encaixa a algum parâmetro faz com que uma entrada seja criada na *cache* do NetFlow, criando assim um registro. O pacote é encaminhado e, pacotes que o sucedem e que também se encaixam nos mesmos parâmetros do primeiro, são agregados ao registro de fluxo recém-criado. Os contadores desse registro são atualizados a cada novo pacote que casa com seu padrão. Os fluxos que estão na *cache* são exportados para um coletor de fluxo (*NetFlow collector*) que, por sua vez, armazena os dados do fluxo em uma base regular. O envio para os coletores de fluxos é realizado por exportadores do NetFlow que enviam registros usando UDP ou SCTP, quando o transporte confiável é necessário. Por fim, os fluxos podem ser analisados por aplicações de análise (*Analysis Applications*). Essas aplicações analisam os dados de fluxo recebidos para fins de detecção de intrusão ou perfil de tráfego. Elas são responsáveis pela apresentação dos dados e criação de relatórios.

Um dos principais pontos fracos relacionado ao NetFlow refere-se à sobrecarga que ele pode impor à infraestrutura de rede. A coleta e o envio dos fluxos pode adicionar sobrecarga aos roteadores e comutadores, por muitas vezes já sobrecarregados. Além disso, quanto mais detalhes os administradores tentam inserir em uma única tupla de fluxo, mais sobrecarga é produzida. Em muitos casos, administradores de rede desabilitam o protocolo para não prejudicar o desempenho das redes. Ressalta-se que o NetFlow tem visibilidade limitada do tráfego, ou pacotes, encaminhados. Como consequência, a comunicação interna (LAN/VLAN) não é contabilizada nos fluxos exportados.

Existe um conjunto de ferramentas alternativas ao NetFlow. São variações que seguem a mesma linha do NetFlow disponibilizadas por fabricantes como Juniper (jFlow), Ericsson (rFlow), Huawei (NetStream) e HP (sFlow). A tecnologia **sFlow**[2], por exemplo, foi desenvolvida pela InMon Inc. que se tornou um padrão da indústria definido na RFC 3176 [Li et al., 2013]. A tecnologia utiliza amostragem aleatória e o agente sFlow é incorporado em comutadores e roteadores de diversos fabricantes. O agente sFlow é um processo de software que associa contadores de interface e amostras de fluxo, gerando datagramas sFlow datagramas que são imediatamente enviados para os coletores sFlow

[2]Coletores para sFlow estão disponíveis em https://sflow.org/products/collectors.php

através de datagramas UDP. Os datagramas sFlow contém informações sobre a versão sFlow, endereço do agente, IP de origem, número de sequência, quantidade de amostras e, normalmente, até 10 amostras de fluxo. A solução de envio imediato de dados minimiza o uso de memória e de processamento. Os pacotes são tipicamente amostrados por Circuitos Integrados Específicos de Aplicação (*Application Specific Circuits* - ASICs) para garantir o alto desempenho de velocidade de fio. Os dados do sFlow contêm o cabeçalho completo do pacote e informações de encaminhamento. O sFlow é capaz de executar em camada 2 e, assim, é capaz capturar tráfego que não é IP. Os coletores sFlow são servidores que coletam os datagramas sFlow. Destaca-se também o **IPFIX** (Internet Protocol Flow Information Export), um padrão derivado do NetFlow v9 para exportar as informações sobre os fluxos de rede. O IPFIX também é capaz de exportar qualquer informação de tráfego de L2-L7 para o coletor de fluxo. É um protocolo flexível que suporta campos de comprimento variável. Ele permite coletar informações como URL ou *host*, como `facebook.com`, bem como os tipos de dados definidos pelo usuário.

4.3. Processamento de Grandes Massas de Dados

O processamento de grandes massas de dados é comumente associado a aplicações de Aprendizado de Máquina (*Machine Learning*). O aprendizado de máquina refere-se à capacidade dos computadores de aprenderem sem serem explicitamente programados [Boutaba et al., 2018] e baseia-se na teoria da probabilidade e estatística, teorema de Bayes e otimização, sendo a base para a análise e a ciência de dados (*Data Science*) no cenário de grandes massas de dados (*Big Data*) [Blei e Smyth, 2017]. A transformação dos dados em informação e conhecimento através, principalmente, das técnicas de aprendizado de máquina é a etapa mais importante de qualquer análise de grandes massas de dados [Hu et al., 2014]. A análise dos dados pode ocorrer em lote ou em fluxo. Contudo, previamente à submissão dos dados para análise é necessário prepará-los. Dessa forma, uma etapa essencial antes da análise é o pré-processamento dos dados.

4.3.1. Análise de dados em lotes (*batch*)

O processamento em lotes é a forma mais tradicional de processamento de dados. Utilizam-se base de dados, as quais registram permanentemente as informações que são posteriormente processadas por meio de consultas (*queries*) às bases. Também são formas comuns o uso de Processamento Analítico Online (*Online Analytical Processing* - OLAP), técnicas de mineração de dados [Carbone et al., 2015b] e processamento paralelo com ferramentas como o Hadoop. No processamento da dados tradicional, utilizam-se operações como seleções relacionais, projeções e agregações, entre outras. Os dados são armazenados em disco e são processados em momentos posteriores à sua coleta, não havendo necessidade de resposta em tempo real, como mostrado na Figura 4.3.

O processamento de dados em lotes é uma forma eficiente de processar grandes volumes de dados quando as transações são coletadas em um período espaçado de tempo. Assim, os dados são coletados, armazenados, processados e, por fim, são gerados os resultados do processamento em lote. A arquitetura e implementação das bases de dados tradicionais, utilizadas no processamento em lotes, não foram projetadas para a inserção e leitura rápida e contínua de dados [Andreoni Lopez et al., 2018]. Contudo, isso não gera problemas, pois as técnicas para processamento em lotes não têm requisitos tão fortes

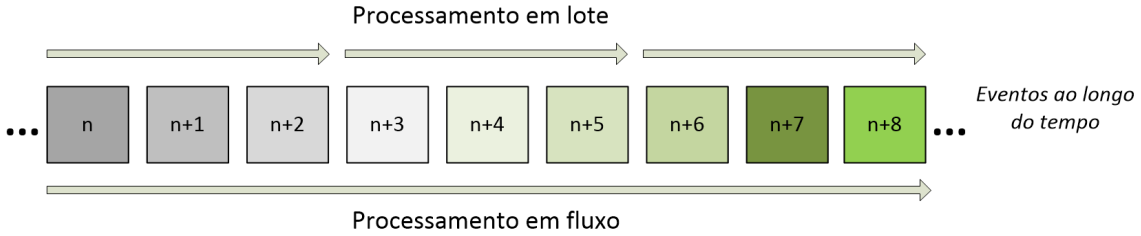


Figura 4.3. Representação do processamento de dados em lote, no qual os dados armazenados em disco são agrupados temporalmente e processados sem necessidade de resposta em tempo real; e em fluxo, no qual os dados em memória são processados assim que recebidos, gerando resposta em tempo real.

com relação à vazão de resultados de processamento. No contexto de controle e monitoramento de redes sem fio, o processamento em lotes é útil, pois dados arquivados são processados gerando informações de interesse que podem ser usadas posteriormente para melhoria da rede.

4.3.2. Análise de dados em fluxo e tempo real

Aplicações que demandam a atuação com baixo tempo de resposta, sejam elas realizadas por um operador ou automaticamente, necessitam combinar a captura de dados em fluxo com o uso de técnicas de inferência e correlação. Devido à alta taxa de chegada de novos dados em fluxo e elevado volume de dados em uma rede de grande porte, os algoritmos de processamento tradicionais carecem de tempo suficiente para percorrer os dados repetidamente de forma iterativa. Para esses cenários, algoritmos de uma única passagem que utilizam pequenas quantidades de memória são necessários, caracterizando a análise de dados em fluxo e tempo real. Essa análise parte do princípio que a massa de dados não é estática e chega constantemente, formando um fluxo de dados, como mostrado na Figura 4.3. Exemplos desse tipo de dados que são produzidos constantemente incluem *logs* de servidores *Web*, *logs* de aplicativos, dados de sensoriamento, mudanças de estado em bases de dados transacionais [Carbone et al., 2015b]. Nesse sentido, o uso do processamento em lotes tradicional torna-se inviável, já que as respostas do processamento só têm validade se respeitarem uma janela de tempo muito curta. Estratégias que fazem o agrupamento de dados por horas, dias ou semanas, para posteriormente processar os dados, geram um atraso intrínseco que pode tornar algumas aplicações específicas inviáveis. Ferramentas para coletas de dados, gerenciadores de fluxo de trabalho e escalonadores funcionam criando um *pipeline* contínuo de lotes [Carbone et al., 2015b].

Embora muitas das informações de interesse de organizações com grandes bases de dados possam ser obtidas com o processamento em lote, algumas aplicações específicas necessitam de um resultado em tempo real da análise dos dados coletados. Isso permite que a organização tome uma atitude imediata em situações que um tempo muito curto pode ser suficiente para causar problemas de diferentes naturezas. Por exemplo, a realização de uma análise para detecção de intrusão demanda uma resposta rápida, para que a intrusão não cause grandes problemas a uma máquina ou à rede. Assim, o principal objetivo do processamento de fluxos em tempo real é conseguir extrair informações para a tomada de decisão em uma janela de tempo muito curta. Ao se realizar uma análise de dados em fluxo e tempo real, os recursos de processamento e memória passam a ser um potencial gargalo, especialmente no contexto de *Big Data*. Os dados chegam em grande

volume, podendo apresentar rajadas, e os resultados devem ser apresentados em tempo real ou quase real. Assim, o uso de memória secundária, em geral, não é possível nesses cenários [Garofalakis et al., 2016], pois o tempo de escrita e de leitura é alto e incompatível com as restrições temporais da aplicação. O processamento em fluxo e tempo real utiliza o esquema "compute primeiro, armazene depois" (*compute-first, store-second*). A necessidade de resposta muito rápida para um grande volume de dados usualmente leva a necessidade do uso de sistemas distribuídos para o processamento da massa de dados.

Outras características também são importantes no processamento em fluxo, quando comparado ao processamento em lote [Andreoni Lopez et al., 2018], tais como, os elementos chegam em linha e devem ser processados tão logo sejam recebidos; o sistema de processamento não tem controle sobre a ordem com que os elementos chegam; os fluxos não têm restrições quanto a tamanho ou duração; e elementos processados são usualmente descartados ou arquivados e, portanto, não são acessados novamente. Os fluxos de dados podem ser representados através de diferentes modelos.

O Modelo de série temporal. Uma série temporal é um conjunto de observações de dados feitas com intervalos de tempo constantes, ao longo de uma janela de tempo. Dada uma observação de uma série temporal, ela pode ser dividida em três componentes básicos, sendo eles: tendência, relacionado a uma visão de longo prazo; sazonalidade, relacionado a movimentos sistemáticos relativos ao calendário; e irregularidade, que são flutuações de curto prazo não sistemáticas. Em uma rede sem fio em uma universidade, por exemplo, a quantidade de usuários em um ponto de acesso costuma sofrer com sazonalidade, pois a quantidade de usuários varia conforme o período de dia e noite, finais de semana e períodos de férias. Usualmente, uma análise de dados realiza um ajuste sazonal, identificando e removendo influências sistemáticas e relacionadas ao calendário. Ainda no exemplo da rede sem fio universitária, os períodos sem usuários devido ao calendário deveriam ser removidos na análise de carga média da rede para dimensionamento e posicionamento dos pontos de acesso. De fato, o processo de ajuste sazonal é importante porque os efeitos sazonais podem mascarar o verdadeiro movimento da série temporal, assim como características não sazonais que sejam de interesse.

O Modelo caixa registradora. Para definir esse modelo, assume-se que o sinal de entrada é representado por um fluxo $a_1, a_2, \ldots$ com chegada sequencial e um sinal A, definido pela função $A[j] : [1 \ldots N] \rightarrow R$. As amostras a_i de entrada são incrementos para o sinal $A[j]$. Por exemplo, seja $a_i = (j, I_i)$, $I_i > 0$ e $A[j]_i = A_{i-1} + I_i$. Nesse caso, A é um fluxo no modelo caixa registradora, em que $A[j]_i$ é o estado do sinal depois da chegada da i-ésima amostra [Lee et al., 2005]. Esse é um modelo de fluxo entre os mais populares, representando, por exemplo, o tempo total de acesso de cada usuário em uma rede após múltiplos acessos, em que j representa cada usuário, I_i é o tempo de cada um de seus acessos e $A[j]_i$ representa o tempo total de acesso do usuário j até o momento i. Outro exemplo constitui o monitoramento de IPs que acessam um ponto de acesso, em que cada IP recebe um valor de j diferente e $A[j]_i$ representa o total de acessos daquele IP.

O Modelo de catraca. Esse modelo é mais genérico que os anteriores, mas é semelhante à caixa registradora. Contudo, no modelo de catraca, I_i pode assumir valores positivos ou negativos. Esse é o modelo mais apropriado para estudar situações amplamente dinâmicas, em que elementos podem ser inseridos ou retirados [Lee et al., 2005].

4.3.3. Técnicas de pré-processamento de dados

Os dados brutos coletados a partir de inúmeras fontes de dados heterogêneas podem ser compostos por uma grande quantidade de dados sem valor ou desnecessários. Isso resulta em conjuntos de dados com níveis de qualidade diferentes, que variam de acordo com a sua completude e intensidade da presença de redundância, ruído e inconsistências [Hu et al., 2014]. Os dados em sua forma bruta, também conhecidos como dados primários [Tsai et al., 2015], apresentam baixa qualidade e ao serem diretamente processados resultam na utilização de um grande espaço de armazenamento e na baixa qualidade da informação gerada. Assim, a qualidade dos dados tem impacto direto no desempenho dos algoritmos de processamento empregados e na qualidade dos resultados obtidos. Dessa forma, uma das primeiras etapas à qual os dados coletados são submetidos antes de serem armazenados para análise posterior é o pré-processamento. O objetivo do pré-processamento é justamente aumentar a qualidade do conjunto de dados obtido e, possivelmente, reduzir o seu volume através da limpeza de inconsistências e ruídos, eliminação da redundância e integração dos dados para fornecer uma visão unificada. Essa etapa pode ocorrer antes ou após a transmissão dos dados, sendo mais adequada a sua execução antes da transmissão, para fins de economia de banda e de espaço de armazenamento [Hu et al., 2014].

Após a realização do pré-processamento, o conjunto de dados obtido pode ser considerado confiável e adequado para a aplicação de algoritmos de mineração de dados [Tsai et al., 2015]. O pré-processamento de dados é de especial importância para as redes sem fio de grande porte devido ao grande volume de dados coletados e à redundância existente neles. Por exemplo, dois pontos de acesso distintos podem coletar dados sobre a rede em uma região onde existe sobreposição de cobertura. Os dados coletados por ambos os pontos de acesso referentes a essa região em comum são fortemente redundantes, sendo desnecessário armazená-los por completo. Além disso, caso fossem processados com a presença da redundância, as conclusões sobre a análise poderiam ser tendenciosas. O pré-processamento inclui diversas técnicas para promover a *limpeza*, a *integração*, a *redução* e a *transformação* dos dados [Hu et al., 2014, García et al., 2016].

Limpeza dos dados. As técnicas para limpeza dos dados determinam a acurácia e a completude dos dados, consertando os problemas encontrados através da remoção ou adição de dados ao conjunto original, e resolvendo inconsistências. Conjuntos de dados reais frequentemente são incompletos devido a falhas no processo de coleta dos dados, limitações no processo de aquisição dos dados ou restrições de custo, que impedem o armazenamento de alguns valores que deveriam estar presentes no conjunto de dados [García et al., 2016]. Além disso, é necessário tratar os erros nos dados, documentando tanto as ocorrências quanto os tipos de erros para que os procedimentos possam ser modificados a fim de reduzir erros futuros [Hu et al., 2014].

Na primeira etapa da limpeza busca-se por dados discrepantes usando diversas ferramentas comerciais de depuração de dados (*data scrubbing tools*) e de auditoria de dados (*data auditing tools*). Ferramentas de depuração de dados usam conhecimentos simples sobre o domínio dos dados para detectar erros e corrigí-los, empregando técnicas de análise (*parsing*) e de correspondência difusa (*fuzzy matching*). Já as ferramentas de auditoria de dados descobrem regras e relações, identificando dados que violam as condi-

ções encontradas. Para tanto, empregam, por exemplo, análise estatística para identificar correlações e algoritmos de agrupamento para identificar *outliers*.

Quando se identifica a ausência de dados, a solução trivial é descartar a instância, mas isso pode resultar na deturpação do processo de aprendizagem e no descarte de informações importantes. Outra abordagem é completar os dados manualmente, o que pode ser impraticável. Uma constante global também pode ser utilizada para completar os dados ausentes, resultando, potencialmente, em uma interpretação errada dos dados. Em vez de uma constante, um valor de tendência central para a característica como a média ou mediana pode ser utilizado. A estratégia mais utilizada é determinar o valor ausente através de modelos probabilísticos aproximados, utilizando procedimentos de máxima verossimilhança [García et al., 2016].

A presença de erros, ou ruído, nos dados é tratada através de duas abordagens principais, polimento de dados (*data polishing*) e filtros de ruído [Liebchen et al., 2007]. O resultado de ambas é semelhante, produzindo uma saída suavizada através de técnicas de regressão, análise de *outliers* e técnicas de categorização (*binning*). Uma sobrecarga computacional extra significativa pode ser adicionada dependendo da complexidade do modelo utilizado. Dessa forma, é necessário buscar o equilíbrio entre o custo adicional da limpeza dos dados e a melhoria da acurácia dos resultados obtidos [Hu et al., 2014].

Integração dos dados. As técnicas de integração de dados, ou resumo, combinam os dados provenientes de diferentes fontes, unificando a visão sobre eles e reduzindo a redundância. A eliminação de redundância também é feita através de técnicas de detecção de redundância e de compressão de dados, permitindo diminuir a sobrecarga na transmissão e no armazenamento dos dados. A redundância pode se apresentar na forma espacial, temporal, estatística ou perceptiva e está fortemente presente em dados de vídeo e imagem. Técnicas para tratamento de redundância nesses casos já são muito bem conhecidas, como JPEG e PNG para imagem e MPEG-2, MPEG-4, H.263 e H.264/AVC para vídeo, mas não são aplicáveis ao contexto de análise de dados em redes sem fio de grande porte. Nesse caso, técnicas de remoção de duplicatas são mais adequadas.

Cada instância pode ser composta por diversas características (atributos) provenientes de fontes diferentes e que podem apresentar esquemas diferentes. Os metadados das características passam a ser importantes para evitar erros durante a integração dos esquemas. Características que podem ser derivadas de outras características ou de um conjunto de características devem ser eliminadas se puderem ser consideradas redundantes. A redundância pode ser avaliada através da análise de correlação, que utiliza o teste *qui-quadrado* (χ^2) para características nominais, e o coeficiente de correlação e covariância para características numéricas. A duplicação de dados também deve ser observada na etapa de integração e a consistência dos dados deve ser garantida, de forma que durante a integração dos dados é necessário detectar e resolver conflitos nos valores das características [García et al., 2016].

Duas estratégias para integração dos dados incluem o uso de Armazém de Dados (*Data Warehouse*) e Federação de Dados (*Data Federation*) [Hu et al., 2014]. No primeiro caso, a integração passa pelas etapas de extração, transformação e carregamento dos dados. Na extração seleciona-se os dados necessários para a análise, que são modificados na etapa de transformação através da aplicação de um conjunto de regras, convertendo

os dados em um formato padrão. Por fim, os dados transformados são importados para a infraestrutura de armazenamento e geralmente são provenientes de uma única fonte [Hu et al., 2014]. No segundo caso, cria-se um banco de dados virtual para consultar e agregar dados de fontes diferentes. O banco de dados virtual contém apenas informação ou metadados sobre os dados reais e sua localização, não armazenando os dados de fato. Essas abordagens são adequadas para processamento de dados em lote, mas ineficientes para dados em fluxo.

Redução dos dados. As técnicas de redução dos dados objetivam diminuir o volume do conjunto de dados, mas de forma que o resultado produzido após o processamento seja mantido. As técnicas incluem estratégias de redução de dimensionalidade e de numerosidade. Também são utilizadas técnicas de compressão de dados para reduzir o volume de dados. A redução de dimensionalidade busca uma representação comprimida dos dados originais, focando na diminuição da quantidade de variáveis aleatórias ou características consideradas. Para tanto são usadas as Transformadas *Wavelet*, a Análise de Componentes Principais (*Principal Component Analysis* - PCA) e Seleção de Características. Também podem ser empregados algoritmos genéticos para reduzir de forma ótima a dimensão do conjunto de dados [Tannahill e Jamshidi, 2014]. Na redução de numerosidade, os dados são substituídos por representações alternativas, de formato menor, usando técnicas paramétricas ou não-paramétricas. As paramétricas, como regressão e modelos log-lineares, estimam os dados e armazenam apenas os parâmetros do modelo que os descrevem. As técnicas não-paramétricas incluem histogramas, algoritmos de agrupamento, técnicas de amostragem e agregação de cubos de dados. Dentre as técnicas de redução, vale destacar as transformadas *wavelet*, o método PCA e a seleção de características.

A *Transformada Wavelet* é uma técnica de processamento de sinais linear em que o sinal passa a ser representado por uma soma de formas de onda mais simples. A transformada *wavelet* tem como objetivo capturar tendências em funções numéricas, decompondo o sinal em um conjunto de coeficientes. Assim, o vetor de dados passa a ser representado por um vetor de coeficientes *wavelet* de mesmo comprimento. Existe uma família de transformadas Wavelet que podem ser usadas no pré-processamento dos dados, sendo as mais populares Haar-2, Daubechies-4 e Daubechcies-6 [Han et al., 2011]. Os coeficientes mais baixos podem ser descartados sem prejuízo significativo para a recuperação dos dados originais. Através do armazenamento de uma pequena fração dos coeficientes de maior intensidade, os dados originais são obtidos aplicando a transformada inversa.

O *PCA* cria um conjunto menor de variáveis que permite representar os dados originais. O PCA é considerado um método de extração de características. Esses dados podem ser descritos através de um vetor de n características ou dimensões e o PCA combina a essência das características originais usando k vetores ortogonais de dimensão n, com $k \leq n$, que melhor representem os dados a serem reduzidos. Como o espaço dimensional de projeção dos dados é menor, a dimensionalidade dos dados é reduzida. Nesse novo espaço dimensional, o PCA pode revelar relações que anteriormente não estavam claras, permitindo interpretações que normalmente seriam ignoradas.

A *Seleção de Características* remove características redundantes ou irrelevantes, mantendo um conjunto mínimo de características que resulte em uma distribuição de probabilidade próxima da distribuição original. A seleção de características reduz o risco de

overfitting dos algoritmos de mineração de dados e reduz o espaço de busca, tornando o processo de aprendizagem mais rápido e consumindo menos memória [García et al., 2016]. Dados representados por um vetor de n características possuem 2^n possíveis subconjuntos, dos quais deve-se escolher uma quantidade ótima para representar os dados. Devido ao grande número de possibilidades que podem ser exploradas para chegar ao resultado ótimo, geralmente são utilizadas heurísticas que exploram um conjunto reduzido do espaço de busca. Geralmente são usados algoritmos gulosos (*greedy*), que escolhem sempre soluções ótimas locais, como seleção passo a passo para frente (*stepwise forward selection*), eliminação passo a passo para trás (*stepwise backward elimination*), uma combinação dos dois anteriores e indução de árvores de decisão. Outra abordagem utiliza o teste de independência qui-quadrado para decidir quais características devem ser selecionadas.

Transformação dos dados: A transformação uniformiza a representação dos dados e geralmente utiliza métodos para reduzir a complexidade e a dimensionalidade do conjunto de dados [Tsai et al., 2015], podendo ser incluída também na etapa de redução dos dados [García et al., 2016]. As principais estratégias para transformar os dados são as técnicas de suavização, construção de características, agregação, normalização, discretização e geração de hierarquias de conceitos para dados nominais. A suavização remove ruído dos dados utilizando, por exemplo, técnicas de regressão e de agrupamento. A construção de características tem como objetivo criar novas características baseadas em outras para melhorar a acurácia e a compreensão de dados de grande dimensionalidade. Juntamente com a seleção de características, a construção de características integra uma área de pesquisa em crescimento conhecida como engenharia de atributos, ou de atributos (*feature engineering*). Os procedimentos de seleção de características identificam e removem o máximo de informação redundante e irrelevante possível [García et al., 2016], enquanto a extração e a construção combinam as características do conjunto original para obter um novo conjunto de variáveis menos redundantes. Na agregação os dados são resumidos, resultando em uma nova representação. O resumo pode ser feito na forma de média, variância, máximo e mínimo. Por exemplo, o consumo diário de energia pode ser agregado para mostrar o consumo mensal máximo, ou anual médio. A normalização modifica a escala dos dados para se ajustarem em uma faixa de valores menor. A discretização substitui a representação de valores numéricos por intervalos numéricos ou rótulos conceituais. Por fim, a geração de hierarquias de conceitos para dados nominais cria múltiplos níveis de granularidade em que os dados podem ser encaixados.

Apesar de a área de pré-processamento de dados já ser bastante explorada, ainda existe muita atividade de pesquisa para buscar novos métodos e aprimorar os métodos existentes, de forma que sejam capazes de lidar com o volume de dados disponíveis cada vez maior e com a geração de conhecimento em tempo real. Existe na literatura trabalhos que estudam novos métodos para, por exemplo, dividir os dados entre processadores em sistemas em nuvem [Ham e Lee, 2014] e realizar seleção de características em dados em fluxo [Andreoni Lopez et al., 2019].

4.3.4. Aprendizado de Máquina

Após a preparação dos dados pelo pré-processamento, a análise de grandes massas de dados transforma os dados em conhecimento usando técnicas de aprendizado de

máquina. O aprendizado de máquina permite inferir soluções para problemas que possam ser representados por um amplo conjunto de dados. As aplicações de aprendizado de máquina são projetadas para identificar e explorar padrões ocultos em dados, para descrever o resultado como um agrupamento de dados em problemas de agrupamento, para prever o resultado de eventos futuros em problemas de classificação e problemas de regressão, e para avaliar o resultado de uma sequência de amostra de dados para problemas de extração de regras [Boutaba et al., 2018]. Assim, de forma simplificada, é possível dividir os problemas de aprendizado de máquina em quatro categorias, agrupamentos (*clustering*), classificação, regressão e extração de regras.

Problemas de agrupamento têm como objetivo minimizar a distância entre amostras de dados com características similares em um mesmo grupo, enquanto maximizam a separação entre grupos distintos. *Problemas de classificação* caracterizam-se por mapearem as entradas em classes-alvos de saída que são representadas pelo conjunto discreto de valores de saída. A coesão dos resultados dos classificadores e dos agrupamentos utiliza frequentemente como parâmetro a Soma dos Erros Quadráticos (*Sum of Squared Errors* - SSE), Equação 1, que leva em consideração o número de agrupamentos, k; a quantidade de dados no i-ésimo agrupamento, n_i; o j-ésimo dado no i-ésimo agrupamento, x_{ij}; a média dos dados no i-ésimo agrupamento, c_i; e a quantidade de dados, n [Tsai et al., 2015]. A Soma dos Erros Quadráticos é dada por

$$SSE = \sum_{i=1}^{k} \sum_{j=1}^{n_i} D(x_{ij} - c_i) \qquad (1) \qquad\qquad c_i = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}. \qquad (2)$$

A métrica de distância, D, mais comum é a distância Euclidiana, definida na Equação 3, onde p_i e p_j são posições de dois dados diferentes. Outras distâncias podem ser usadas, como as distâncias Manhattan e Mikowski.

$$D(P-i, p_j) = (\sum_{l=1}^{d} |p_{il}, p_{jl}|^2)^{\frac{1}{2}} \qquad (3)$$

Métricas usadas para avaliar os resultados da classificação são a acurácia (Equação 4), a precisão (Equação 5), a revocação (Equação 6), a especificidade 7 e F-measure (Equação 8), que permitem descobrir os acertos (verdadeiros positivos, VP, e verdadeiros negativos, VN) e as falhas na classificação, como a quantidade de dados que não pertencem ao grupo mas são incorretamente classificadas como pertencentes (falsos positivos, FP) e a quantidade de dados que pertencem ao grupo que não são classificados como pertencente ao grupo (falsos negativos, FN). Essa interpretação pode ser sumarizada através de uma matriz de confusão do classificador, conforme indicado na Tabela 4.1. Outra métrica importante na avaliação de classificadores é a Área Abaixo da Curva ROC (*Receiver Operating Characteristic* - ROC), conhecida como AUC (*Area Under the ROC Curve*), que permite verificar o compromisso entre a taxa de verdadeiros positivos e a taxa de falsos positivos. O tamanho da AUC é diretamente proporcional ao desempenho do classificador. Andreoni Lopez et al. [Andreoni Lopez et al., 2018] explicam detalhadamente cada uma dessas métricas.

	Classificação positiva	Classificação negativa
Positivo (P)	Verdadeiro positivo (VP)	Falso negativo (FN)
Negativo (N)	Falso positivo (FP)	Verdadeiro negativo (VN)

Tabela 4.1. Matriz de confusão

$$acc = \frac{VP+VN}{P+N}. \quad (4) \qquad p = \frac{VP}{VP+FP}. \quad (5) \qquad r = \frac{VP}{VP+FN}. \quad (6)$$

$$e = \frac{VN}{FP+VN}. \quad (7) \qquad F = \frac{2pr}{p+r}. \quad (8)$$

Os *problemas de regressão* tendem a gerar modelos matemáticos que tendem a se comportar como funções multivariáveis em que os valores de entrada são mapeados em valores contínuos de saída. Já os *problemas de extração* de regras são diferentes dos demais, pois o objetivo desse problema não é inferir valores de saída para cada conjunto de entrada, mas identificar relações estatísticas entre os dados.

Os principais paradigmas de aprendizagem de máquina são supervisionados, não-supervisionados, semi-supervisionados e aprendizado por reforço (*reinforcement learning*). A definição do paradigma de aprendizagem a ser usado em uma aplicação de aprendizado de máquina determina como os dados devem ser coletados, o estabelecimento da verdade básica responsável pela inserção de rótulos dos dados e a engenharia de características responsável por estabelecer as características dos dados e quais delas devem ser exploradas na aplicação. A Figura 4.4 mostra o processamento em fluxo e em lote com processos que representam os quatro paradigmas de aprendizado de máquina. O *aprendizado supervisionado* baseia-se em um conjunto de dados rotulados, chamado de conjunto de treinamento, para criar o modelo de classificação ou regressão dos dados. Esse paradigma de aprendizagem requer a existência *a priori* de um conjunto de

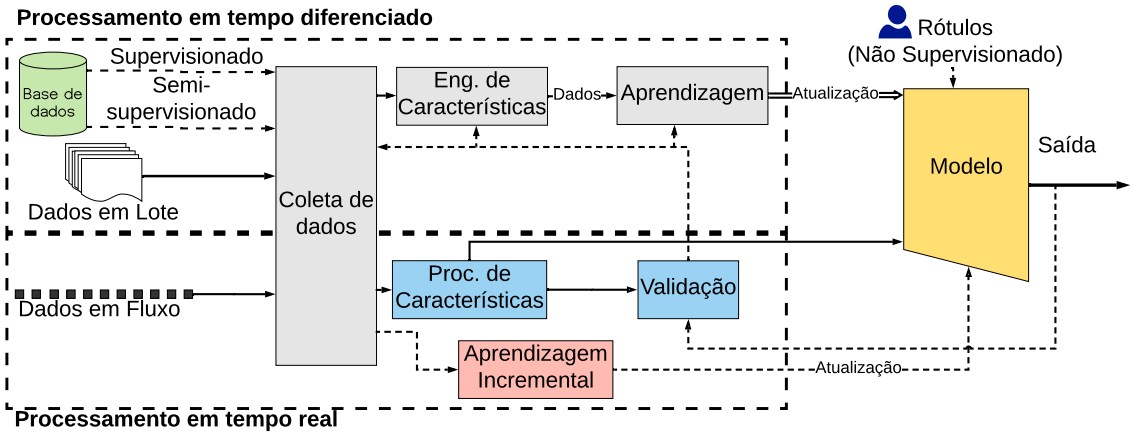


Figura 4.4. Representação dos quatro paradigmas de aprendizado de máquina em um sistema de processamento de dados em lote e em fluxo. A característica contínua e ilimitada dos dados viabiliza o aprendizado incremental, em que o modelo aprende com os dados entrantes. Já o processamento em lote, o treinamento do modelo é somente realizado com bases históricas armazenadas.

dados rotulados para a criação do modelo. Por sua vez, o paradigma de *aprendizado semi-supervisionado* suporta o uso de conjuntos de treinamento com rótulos faltantes ou incompletos. Já o *aprendizado não-supervisionado* busca padrões nos dados de treinamento e, portanto, não requer a existência prévia de dados rotulados. O aprendizado não-supervisionado é adequado para problemas de agrupamentos (*clustering*), em que se busca um padrão de agrupamento dos dados para maximizar a distância entre grupos, enquanto se minimiza a distância entre dados dentro de um mesmo grupo. Por fim, o paradigma de *aprendizado por reforço* consiste em um processo iterativo, em que agentes aprendem efetivamente novos conhecimentos na ausência de qualquer modelo explícito do sistema, com pouco ou nenhum conhecimento do ambiente [Tesauro, 2007]. A ideia central do aprendizado por reforço é que o aprendizado de um agente é baseado em exemplos do conjunto de treinamento, que interage com o mundo externo e aprende com reforços providos pelo ambiente. Os reforços providos podem ser recompensas ou penalidades. Assim, o conjunto de treinamento consiste de pares de amostras de dados e reforços, sejam recompensas ou penalidades. A retroalimentação do ambiente impulsiona o agente para a melhor sequência de ações. O aprendizado por reforço é adequado para problemas de tomada de decisão, planejamento e programação [Tesauro, 2007].

Estabelecer a verdade básica consiste na descrição formal, rótulos, dada às classes de interesse. Os métodos para rotular conjuntos de dados usando as características de uma classe são variados. O método mais elementar é a rotulagem manual por especialistas, com auxílio de ferramentas específicas que, por exemplo, realizam a inspeção profunda de pacotes (*Deep Packet Inspection* – DPI) [Andreoni Lopez et al., 2019] , correspondência de padrões como em sistemas de detecção de intrusão ou técnicas não-supervisionadas como o agrupamento de dados brutos [Zhang et al., 2016]. Uma alternativa a modelos manuais é o desenvolvimento de modelos estatísticos e estruturais que descrevem os dados para inferir uma classe de interesse. Contudo, a definição da verdade básica no estabelecimento do conjunto de treinamento está estreitamente relacionada à acurácia e à precisão do modelo de aprendizado de máquina. A dependência mútua inerente do tamanho dos dados de treinamento de uma classe de interesse em relação às outras impacta o desempenho do modelo [Andreoni Lopez et al., 2018]. Muitas técnicas de aprendizado de máquina assumem que as classes de interesse têm distribuição semelhante no conjunto de treinamento.

O aprendizado de máquina é uma ferramenta poderosa para obter conhecimento a partir de dados coletados em um determinado cenário. Dados provenientes de redes sem fio podem trazer diversas informações escondidas. Especificamente para redes Wi-Fi, através da coleta e análise dos dados é possível descobrir perfis de uso da rede [Afanasyev et al., 2010, Fan et al., 2016], determinar a localização dos usuários [Huang et al., 2016] e monitorar o deslocamento [Chilipirea et al., 2016], permitindo inclusive descobrir padrões de comportamento na movimentação dos usuários [Zeng et al., 2015], reconhecimento de atividades [Alsheikh et al., 2016] e identificação do próprio usuário [Zeng et al., 2016].

4.4. Ferramentas para Processamento de Grandes Massas de Dados

Em redes sem fio, o modelo ideal para processamento de grandes massa de dados é o processamento em fluxo, devido à característica dinâmica e ao volume de dados não limitado gerado pelas redes sem fio. Assim, são originadas diversas oportunidades para

a indústria e a pesquisa gerarem conhecimento e inteligência. No entanto, esse grande volume de dados também traz diversos desafios computacionais, já que a capacidade de processamento de uma única máquina não acompanhou o crescimento do volume de dados. Com isso, surge a necessidade de um sistema distribuído de processamento [Zaharia et al., 2012b]. Nesse contexto, inúmeros modelos de programação de aglomerados computacionais (*clusters*) surgiram visando o tratamento de diversas cargas de trabalho [Low et al., 2012, Isard et al., 2007, Dean e Ghemawat, 2008].

O processamento em fluxo (*stream processing*) possibilita a análise de dados em tempo real, como a análise de arquivos de *logs* de servidores em tempo real ou análise de fluxos de rede a medida que os pacotes chegam aos dispositivos. Ferramentas de processamento de dados em lote são utilizadas para trabalhar com grandes massas de dados estáticos, a ferramenta mais utilizada de processamento de dados em lote é o Hadoop [Dean e Ghemawat, 2008]. Para análise de dados em redes é necessária uma ferramenta de processamento de dados em fluxo, pois os fluxos de rede são dinâmicos. Existem diversas propostas de plataformas para processamento em fluxo como Apache Spark Streaming [Zaharia et al., 2012b], Apache Storm [Iqbal e Soomro, 2015] e Apache Flink [Carbone et al., 2015a]. O Apache Storm é bastante utilizado, mas o Flink possui um desempenho melhor quanto ao processamento em fluxo e o Spark possui um sistema de recuperação de falhas superior [Andreoni Lopez et al., 2016, Chintapalli et al., 2016]. O Apache Hadoop, adotado no processamento em lote e usado como base para outras ferramentas como o Apache Spark, e o Apache Storm são explorados em trabalhos anteriores [Andreoni Lopez et al., 2018].

4.4.1. Apache Spark

Uma ferramenta de processamento distribuído de dados em fluxo amplamente utilizada no mercado é o Apache Spark Streaming. O Spark é uma ferramenta de processamento distribuído em lote que possui extensões que dão suporte a diversas cargas de trabalho. O Spark possui extensões para SQL, aprendizado de máquina, processamento de gráficos e processamento em fluxo, utilizando o conceito de microlotes (*microbatch*) [Zaharia et al., 2012b]. Possui também os módulos YARN e Mesos para a gerência dos aglomerados computacionais (*clusters*), permitindo que o usuário foque na programação de novos aplicativos sem se preocupar com a localização dos dados ou em qual nó o dado será processado. A generalidade de propósito do Spark contribui para diversos benefícios: (i) facilidade no desenvolvimento de aplicações, pois o Spark possui uma API unificada; (ii) maior eficiência na execução de tarefas de processamento e no armazenamento em memória, pois o Spark pode executar diversas funções sobre os mesmos dados, geralmente na memória, enquanto outros sistemas exigem a gravação de dados em armazenamento não volátil para que outros mecanismos os acessem; (iii) possibilidade de criação de novas aplicações, como *queries* interativas em gráficos e aprendizado de máquina em linha (*online*), o que não é possível em outras ferramentas.

Para obter resiliência no armazenamento dos dados, o Spark pode utilizar o HDFS ou o serviço simples de armazenamento (*Simple Storage Service* - S3), que são sistemas de arquivos para dividir, espalhar, replicar e gerenciar dados nos nós de um aglomerado computacional [Andreoni Lopez et al., 2018]. Com isso, os dados armazenados em disco serão distribuídos em diversos nós do aglomerado computacional, assim como os dados

processados em memória, criando um ecossistema tolerante a falhas. O compartilhamento de dados é a principal diferença entre o Spark e outras plataformas do paradigma *MapReduce*, tornando o Spark mais rápido que plataformas anteriores para consultas interativas e algoritmos iterativos, como aprendizado de máquina [Xin et al., 2013]. O Spark possui um paradigma de programação similar ao *MapReduce* do Hadoop, porém possui sua própria abstração de compartilhamento de dados, o Conjunto de Dados Distribuídos Resilientes *(Resilient Distributed Dataset* - RDD). O RDD é a principal estrutura de dados do Spark e é responsável por armazenar os dados em memória e distribuí-los garantindo a resiliência [Andreoni Lopez et al., 2018]. Para cada transformação de dados, como `map`, `filter` e `groupBy` é criado um novo RDD. A manipulação dos RDDs é feita por meio de uma API que pode ser utilizada pelas linguagens Scala, Java, Python e R, em que os usuários podem simplesmente passar funções locais para serem executadas no aglomerado computacional.

O Código 4.1, escrito em Python, demonstra a criação de um RDD que lê mensagens de erro dentro de um arquivo de *log* armazenado em HDFS. O Spark percorre todas as linhas do arquivo em tempo real, mesmo que o servidor esteja gravando novos dados no fim do arquivo, e cria outro RDD somente com as linhas que contêm a palavra inicial "*ERROR*". A 5ª linha define um RDD criado a partir do arquivo de *log* armazenado no HDFS, mediante a coleta de linhas no arquivo. A 6ª linha chama a transformação `filter` para derivar um novo RDD a partir do RDD *log*. A última linha chama a função `count`, outro tipo de operação sobre RDD chamada de ação (*action*), que retorna um resultado, não outro RDD, nesse caso, o número de elementos em um RDD.

```
1 from pyspark import SparkContext, SparkConf
2
3 conf = SparkConf().setAppName("ErrorCount").setMaster("local[*]")
4 sc = SparkContext(conf=conf)
5 log = sc.textFile("hdfs://...")
6 erros = lines.filter(lambda x: x.startswith('ERROR'))
7 print('Total de erros: s%' % errors.count())
```

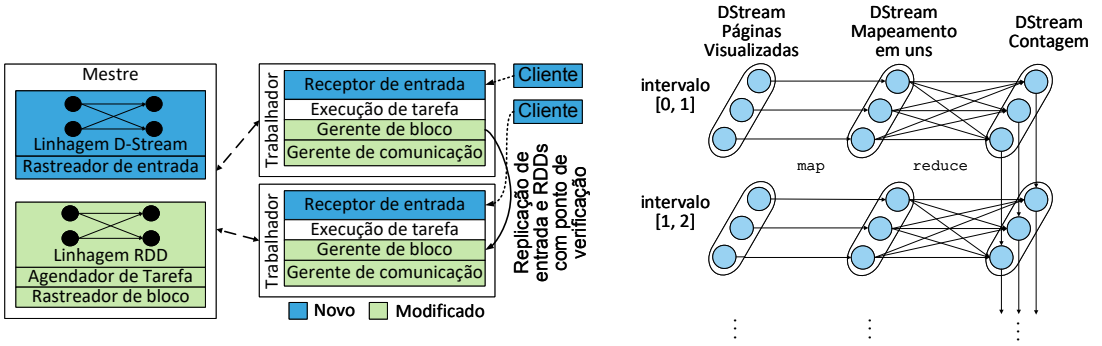
Listing 4.1. Código de leitura de Log utilizando a API pySpark

Sistemas de computação distribuída costumam prover tolerância a falha através de replicação de dados ou criando pontos de verificação (*checkpoints*). Já o Spark utiliza uma abordagem diferente chamada linhagem (*lineage*). Cada RDD grava um grafo de todas as transformações que foram utilizadas para construí-lo e retorna essas operações para uma base de dados para uma futura reconstrução no caso de perda de dados. Recuperação baseada em linhagem é significantemente mais eficiente do que replicação, obtendo um ganho de tempo, pois transferir dados pela rede é muito mais lento do que a escrita na memória RAM e no armazenamento. A recuperação é muito mais rápida do que simplesmente executar novamente o programa, pois o nó com falha geralmente possui múltiplas partições RDDs e essas partições podem ser reconstruídas em paralelo com outros nós [Zaharia et al., 2012a]. O Apache Spark possui quatro bibliotecas de alto nível que fazem o uso da estrutura RDD para uma variedade de funções. A **Spark SQL** implementa *queries* em linguagem de consulta estruturada (*Structured Query Language* - SQL) em banco de dados relacional. A ideia principal dessa biblioteca é ter o mesmo *layout* de banco de dados analíticos dentro dos RDDs. O Spark SQL provê uma abstração

de alto nível para transformações de dados chamadas de *DataFrames*, que são abstrações comuns para dados tubulares em Python e R, com métodos programáticos para filtros e agregações [Armbrust et al., 2015]. A **GraphX** provê uma interface de computação de operações sobre grafos similar ao Pregel [Malewicz et al., 2010] e GraphLab [Low et al., 2014] e implementa a otimização de posicionamento desses sistemas por meio de sua opção de função de particionamento para o RDD [Gonzalez et al., 2014]. A **MLlib** é a biblioteca responsável pelas funções de aprendizado de máquina, possuindo algoritmos de classificação, regressão, agrupamentos, filtros colaborativos, engenharia de características, construção de *pipelines*, e outras funções. O MLlib permite que algoritmos de aprendizado de máquina operem de forma distribuída em aglomerados de computadores. O MLlib foi escrito em Scala e usa bibliotecas nativas de álgebra linear baseadas em C++. MLlib possui APIs para Java, Scala e Python e é distribuído como parte do projeto Spark [Meng et al., 2016]. MLlib possui uma documentação rica, descrevendo todos os métodos suportados e códigos de exemplo para cada linguagem suportada.

A quarta biblioteca de alto nível do Spark o habilita para o processamento de dados em fluxo. O **Spark Streaming** implementa um modelo de processamento de fluxo discretizado (*Discretized Streaming* - D-Streams), que viabiliza a tolerância a falhas e trata os dados retardatários introduzidos por nós mais lentos [Zaharia et al., 2013]. O Spark Streaming é composto por três componentes. O **Mestre** que monitora o grafo de linhagem do D-Stream e organiza as amostras para calcular novas partições RDD. Os **Nós trabalhadores**, que recebem e processam os dados, também armazenam as partições de entrada e calculam os RDDs. O **Cliente** que envia os dados para a ferramenta. O Spark Streaming modifica e adiciona vários componentes ao Spark para ativar o processamento em fluxo. A Figura 4.5(a) mostra os componentes novos e modificados pelo Spark Streaming. A comunicação de rede, por exemplo, é reescrita para permitir que tarefas com entradas remotas sejam recuperadas rapidamente. Existem modificações no mecanismo de linhagem, como remoção de grafos de linhagens que já possuem ponto de verificação (*checkpoint*), assim evitando que os grafos cresçam indefinidamente. Do ponto de vista arquitetural, o que diferencia o Spark Streaming de outras ferramentas que operam sobre dados em fluxo é a divisão das tarefas computacionais em instâncias determinísticas, curtas, sem estado, cada uma delas podendo ser executada em qualquer nó do aglomerado computacional, ou até mesmo em vários nós. O D-Stream discretiza um fluxo de dados em pequenos lotes chamados de microlotes para que eles possam ser tratados pelas tarefas computacionais adequadas.

As aplicações de *big data* costumam utilizar duas abordagens para lidar com tolerância a falha: replicação e *upstream backup*. Em replicação há duas cópias dos dados processados e registros de entrada são replicados para outros nós, porém somente a replicação não é suficiente. É necessário também um protocolo de sincronização para garantir que as cópias de cada operador vejam os dados herdados na mesma ordem. Replicação é custosa computacionalmente, embora a recuperação de uma falha seja rápida. Já em *upstream backup* cada nó retem uma cópia do dado enviado desde um determinado ponto de verificação. Quando um nó falha, uma máquina em modo de espera assume o seu papel e os nós pais reproduzem as mensagens a essa máquina, que antes estava em modo de espera, para que ela mude o seu estado. Ambas as abordagens possuem desvantagens. Replicação tem um grande consumo de recursos de hardware e *upstream*



(a) Arquitetura do Spark Streaming, mostrando seus componentes. Adaptado de [Zaharia et al., 2013]

(b) Grafo de linhagem de um programa de contagem de visualização de páginas WEB. Adaptado de [Zaharia et al., 2013]

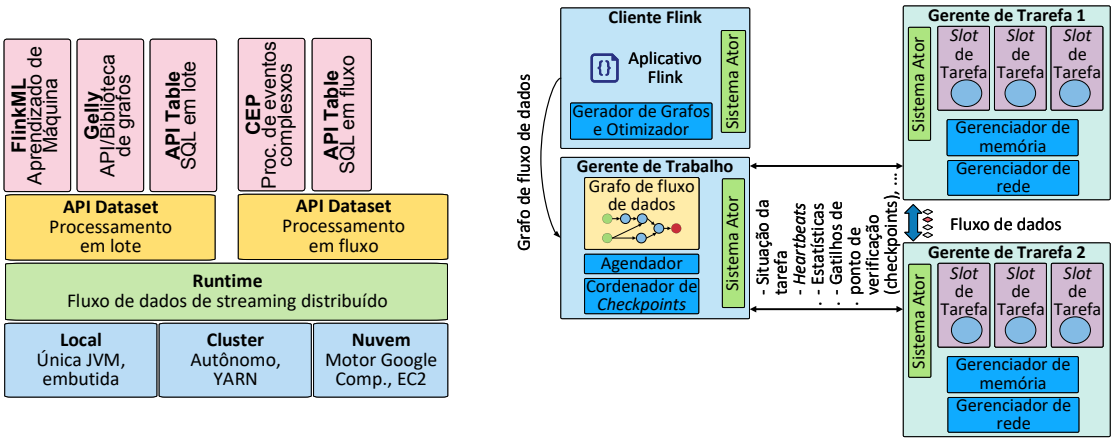
Figura 4.5. Arquitetura do Apache Streaming e grafo de linhagem de um D-Stream.

backup é lento no processo de recuperação. Ambas as abordagens não conseguem lidar com dados retardatários. A ideia do D-Stream é computar os dados em fluxo sem estado (*stateless*) utilizando computação em lote determinística em intervalos curtos de tempo. O D-Stream estrutura as computações da seguinte maneira: i) o estado em cada etapa temporal é totalmente determinístico ao dado de entrada, abolindo a necessidade de protocolos de sincronização e ii) a dependência entre o estado atual e o antigo são visíveis em uma fina granularidade. Isso garante ao Spark Streaming um excelente mecanismo de recuperação, similar aos de sistemas em lote e que supera a replicação e *upstream backup*. A latência do D-Stream é baixa, pois ele usa a estrutura de dados RDD, que computa os dados na memória e utiliza a abordagem de linhagem, conforme ilustrado na Figura 4.5(b). O D-Stream possui uma rápida recuperação de falhas e dados retardatários, pois utiliza o determinismo para prover um novo mecanismo de recuperação que é a recuperação paralela. Quando um nó falha, cada nó do *cluster* trabalha para computar novamente e recuperar os RDDs do nó em falha, resultando em uma recuperação significantemente mais rápida do que as outras duas abordagens [Zaharia et al., 2013]. O D-Stream pode recuperar retardatários utilizando execução especulativa [Dean e Ghemawat, 2008]. Cada D-Stream, assim como os RDDs, são imutáveis, assim cada transformação gera um novo D-Stream. Um D-Stream é uma sequencia de conjunto de dados imutáveis e particionadas (RDD) que podem ser influenciados pela transformação determinística.

4.4.2. Apache Flink

O Apache Flink [Carbone et al., 2015a] foi proposto em 2009 com o nome Stratosphere [Warneke e Kao, 2009]. Foi incubado na fundação Apache em 2014 e em dezembro do mesmo ano foi promovido a projeto *Top Level*. O Flink suporta processamento de fluxo e lote, tornando-se uma plataforma de processamento híbrida. A arquitetura do Flink pode ser distinguida em quatro camadas, Desenvolvimento, núcleo, APIs e bibliotecas, conforme apresentados na Figura 4.6(a).

O núcleo do Flink é o motor distribuído de fluxo de dados, que executam os programas desenvolvidos. As tarefas de analítica do Flink são abstraídas em grafos acíclico dirigido (GAD) formado por quatro componentes: fontes; operadores; torneiras de saída; e registros que percorrem o grafo. O grafo apresentado na Figura 4.7, consiste em: i) ope-



(a) Arquitetura do Flink. Adaptado [Carbone et al., 2015a]

(b) Modelo de processamento do Flink. Adaptado [Carbone et al., 2015a]

Figura 4.6. Arquitetura e modo de trabalho do Flink

radores de estado (*stateful*) e ii) fluxos de dados que representam dados produzidos por um operador e estão disponíveis para consumo pelos operadores. Como os grafos de fluxo de dados são executados de forma paralela, os operadores são paralelizados em uma ou mais instâncias, chamadas subtarefas, e fluxos são divididos em uma ou mais partições. Os operadores de estado, que podem ser sem estado, em casos especiais, implementam toda a lógica de processamento, como operações de `filter`, `map`, `hash join`, `window` etc. Os fluxos distribuem dados entre operadores de produção e consumo em vários padrões, como ponto a ponto, propagação (*broadcast*), repartição, difusão (*fan-out*) e mesclagem.

O Flink possui duas APIs bases, a API DataSet para processamento de dados estáticos e finitos, frequentemente associados a processamento em lote, e a API DataStream para processamento de dados dinâmicos e potencialmente ilimitados, frequentemente associada a processamento em fluxo. As bibliotecas do Flink são específicas para cada API e são FlinkML para aprendizado de máquina, Gelly para processamento de grafos e Table para processamento de operações SQL.

Um cluster do Flink trabalha no modelo mestre-escravo conforme mostrado na Figura 4.6(b) e é composto por três tipos de processos: o cliente, o Gerente de Trabalhos e pelo menos um Gerente de Tarefas. O cliente pega o código do programa Flink, o transforma em um gráfico de fluxo de dados e o envia para o gerente de trabalho. O gerente de trabalho coordena a execução distribuída do fluxo de dados. Ele rastreia o estado e o progresso de cada operador e fluxo, programa novos operadores e coordena os pontos de verificação (*checkpoints*) e a recuperação. Em uma configuração de alta disponibilidade, o gerente de trabalho persiste em um conjunto mínimo de metadados em cada ponto de verificação para um armazenamento tolerante a falhas, de modo que um gerente de trabalho em espera possa reconstruir o ponto de verificação e recuperar a execução do fluxo de dados a partir dele. O processamento de dados ocorre nos gerentes de tarefas. Um Gerente de tarefas executa um ou mais operadores que produzem fluxos e relatórios sobre seu status para o gerente de trabalho. Os gerentes de tarefas mantêm os *buffer* para armazenar ou materializar os fluxos e as conexões de rede para trocar os fluxos de dados entre os operadores [Carbone et al., 2015a]. A abstração do fluxo de

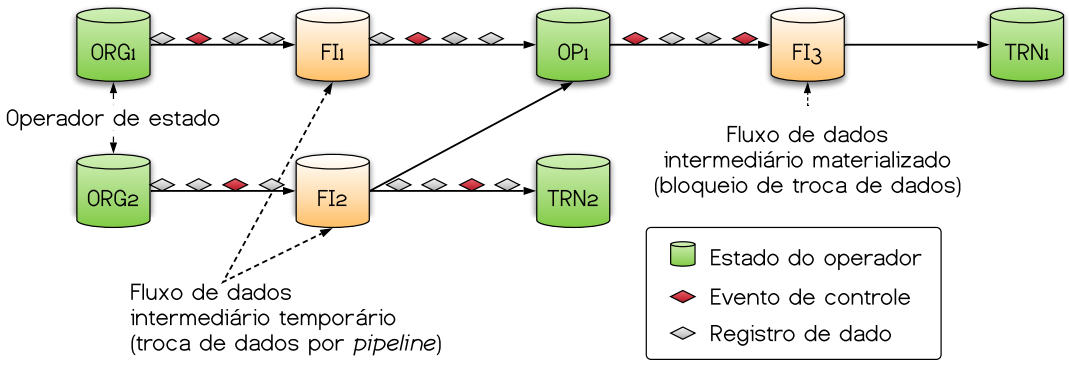


Figura 4.7. Grafo de fluxo de dados. Adaptado [Carbone et al., 2015a]

dados no Flink é chamada DataStream e é definida como uma sequência de registros parcialmente ordenados. Parcialmente, pois não há garantia de ordem caso um elemento operador receba mais de um fluxo de dados como entrada [Andreoni Lopez et al., 2018].

O Flink oferece execução confiável com garantias de consistência da semântica de processamento "exatamente uma" e lida com falhas com ponto de verificação e re-execução parcial. O mecanismo de verificação do Apache Flink baseia-se em captura momentânea do estado do sistema (*snapshot*) distribuídos para obter garantias de processamento exatamente uma vez. A natureza possivelmente ilimitada de um fluxo de dados torna a recomputação na recuperação impraticável, já que possivelmente meses de computação precisarão ser repetidos para uma tarefa longa. Para diminuir o tempo de recuperação, o Flink obtém uma captura de estado *snapshot* dos operadores, incluindo a posição atual dos fluxos de entrada em intervalos regulares. O mecanismo usado no Flink é chamado de Captura de estado de Barreiras Assíncronas (*Asynchronous Barrier Snapshotting* - ABS) [Carbone et al., 2015a]. As barreiras são registros de controle injetados nos fluxos de entrada que correspondem a um tempo lógico e separam logicamente o fluxo para a parte cujos efeitos serão incluídos na captura atual e a parte que será capturada posteriormente. Um operador recebe barreiras de *upstream* e primeiro executa uma fase de alinhamento, certificando-se de que as barreiras de todas as entradas foram recebidas. Então o operador escreve seu estado, como por exemplo conteúdos de uma janela deslizante ou uma estrutura de dados customizada em um sistema de arquivo durável, com o HDFS por exemplo. A recuperação de falhas reverte todos os estados do operador para seus respectivos estados retirados da última captura de estado bem-sucedida e reinicia os fluxos de entrada a partir da última barreira para a qual há uma captura de estado. A quantidade máxima de recálculo necessária na recuperação é limitada à quantidade de registros de entrada entre duas barreiras consecutivas. Além disso, a recuperação parcial de uma subtarefa com falha é possível, reproduzindo registros não processados armazenados em *buffer* nas subtarefas do fluxo ascendente.

4.5. Processamento de Grandes Massas de Dados em Fluxo

O modelo de processamento de fluxos prevê os dados chegando em linha (*online*) e, com isso, o sistema não possui o controle da ordem de chegada dos dados e, nem mesmo, da frequência com que os dados chegam. Os dados chegam de maneira contínua e ilimitada, sendo que o tamanho e tipo dos dados são desconhecidos. No processamento

de fluxos em tempo real a amostra é processada apenas uma vez, visto que o sistema tem de estar disponível a novos dados [Andreoni Lopez et al., 2018].

Amostras podem ser armazenadas temporariamente em memória, mas a memória é pequena em relação ao tamanho potencial dos dados que chegam nos fluxos. Dessa forma, para atingir o objetivo do processamento dos dados em tempo real, o processamento por fluxo impõe restrições ao tempo de processamento por amostra, já que se a taxa de chegada de dados for maior que a taxa de serviço de processamento, a fila de espera de dados a processar cresce indefinidamente e, como consequência, haveria descarte de dados. Portanto, técnicas para prover o processamento eficiente por fluxo constituem o uso de plataformas de processamento distribuído e o uso de técnicas de aproximação para agilizar o processamento dos dados.

Os métodos tradicionais de aprendizado de máquina se baseiam em sistemas que assumem que todos os dados coletados são totalmente carregados em um lote único, para então serem processados de maneira centralizada [Qiu et al., 2016]. Conforme ocorre o aumento do volume de dados, as técnicas existentes de aprendizado de máquina falham ao serem confrontadas com volumes inesperados de dados e, também, com o requisito de retornar a saída do processamento com a mesma agilidade que os dados são gerados. Assim, surge a necessidade de se desenvolver novos métodos de aprendizado de máquina mais rápidos e com comportamento adaptativo para atender às demandas do processamento de grandes massas de dados em tempo real [Costa et al., 2012].

Em muitos casos, certos padrões e rastros de comportamento estão escondidos no meio de um grande volume de informação e só são descobertos com auxílio de sistemas que utilizam aprendizado de máquina. Conforme novas informações são disponibilizadas, as estruturas de decisão devem ser revistas e atualizadas. Diversos modelos atualizam seus parâmetros considerando uma amostra por vez. Esses modelos são: aprendizado incremental, aprendizado *online*, aprendizado sequencial ou revisão da teoria [Gama et al., 2014]. Métodos incrementais não possuem restrição de tempo ou ordem das amostras, enquanto métodos *online* exigem que as amostras sejam processadas em ordem e apenas uma vez, de acordo com o tempo de chegada. Muitos algoritmos incrementais podem ser usados de maneira *online*, mas algoritmos que modelam comportamento no tempo necessitam que as amostras estejam em ordem.

Em ambientes dinâmicos e não estacionários, a distribuição de dados pode mudar ao longo do tempo, produzindo o fenômeno do desvio de conceito (*Concept Drift*). O desvio de conceito refere-se a mudanças na distribuição condicional da saída, ou seja, variável alvo, dado o vetor de características de entrada, enquanto a distribuição da entrada pode permanecer inalterada [Gama et al., 2014]. Um exemplo de mudança de conceito são os padrões de consumo de clientes. As preferências de compras podem mudar com o tempo, dependendo do dia da semana, disponibilidade de alternativas ou mudanças de salario. Em segurança, a criação de modelos para a detecção de ameaças pode ficar obsoleta com mínimas variações na composição dos ataques [Lobato et al., 2018]. A mudança de conceito afeta o desempenho da maioria dos algoritmos de aprendizado tornando-os menos precisos à medida que o tempo passa.

Um preditor eficaz deve ser capaz de rastrear essas mudanças e se adaptar rapidamente a elas. Um problema difícil ao lidar com a mudança de conceito é distinguir

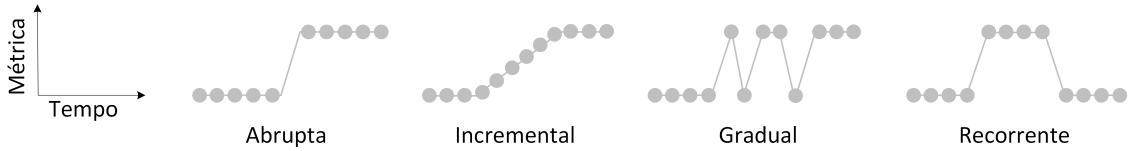


Figura 4.8. Tipos de mudanças de conceito. Em ambientes dinâmicos a distribuição de dados pode mudar ao longo do tempo, produzindo o fenômeno de mudança de conceito (*Concept Drift*).

entre uma mudança real e ruído. Alguns algoritmos podem reagir excessivamente ao ruído, interpretando-o erroneamente como mudança de conceito, enquanto outros podem ser altamente robustos ao ruído, ajustando-se às mudanças muito lentamente. Os quatro tipos diferentes de mudanças conhecidas são (Figura 4.8): i) mudança súbita ou abrupta; ii) mudança incremental; iii) mudança gradual; e iv) mudança recorrente ou cíclica.

Diferentes abordagens na detecção de mudança de conceito podem ser usadas dependo do domínio da classificação [Andreoni Lopez et al., 2019]. A primeira e a mais simples assume que os dados são estáticos e, portanto, não existe mudança na distribuição dos dados, treinando o modelo uma única vez e usando o mesmo modelo para dados futuros. Outra abordagem é atualizar periodicamente o modelo estático com dados históricos mais recentes, também conhecido como aprendizagem incremental. Alguns algoritmos de aprendizado de máquina como os algoritmos de regressão ou as redes neurais permitem avaliar a importância dos dados de entrada. Nesses algoritmos, é possível usar uma ponderação inversamente proporcional à história dos dados, de modo que seja dada maior importância aos dados mais recentes, com um peso maior, e menor aos dados menos recentes, com peso menor. Outra abordagem é o uso de algoritmos de conjunto de classificadores como AdaBoost ou Floresta Aleatória (*Random Forest*). Assim, o modelo estático é deixado intacto, mas um novo modelo aprende a corrigir as previsões do modelo estático com base nos relacionamentos de dados mais recentes. Finalmente, também é possível detectar mudanças de conceito utilizando heurísticas ou estatísticas intrínsecas do dados. Heurísticas como a acurácia ou a precisão são principalmente utilizadas em um cenário de aprendizado supervisionado no qual as etiquetas dos dados estão presentes durante o treinamento e a classificação. No entanto, a presença de etiquetas durante a classificação não é habitual em um ambiente de produção. No cenário de aprendizado não supervisionado, a comparação estatística de amostras de chegada, ou o agrupamento de amostras, com as amostras usadas para treinar o sistema, pressupõe que uma mudança de conceito é detectada sempre que novos grupos são encontrados [Jordaney et al., 2017]. Esses métodos de detecção de mudanças são propícios ao consumo de recursos, já que as medidas baseadas em distâncias são realizadas sobre as amostras obtidas.

4.5.1. Técnicas para a Mineração de Dados em Fluxo

Gaber *et al.* categorizam as soluções de tratamento de fluxos como baseadas em dados ou baseadas em tarefas [Gaber, 2012]. As soluções baseadas em dados têm como objetivo diminuir a representação dos dados, seja através de transformações horizontais, diminuindo o número de características a serem tratadas, ou transformações verticais, selecionando-se um subgrupo de observações a serem tratadas, também conhecido como amostragem. As soluções baseadas em tarefas concentram-se em uso de técnicas computacionais para tornarem as soluções eficientes tanto em relação a tempo quanto a espaço

de armazenamento. As técnicas baseadas em dados baseiam-se em resumir os dados ou escolher subconjuntos de dados do fluxo de entrada.

Amostragem de dados (*Sampling*). Enquanto algumas tuplas de dados são selecionadas para o processamento, outras são descartadas. No caso de chegada de dados a uma taxa superior ao que o sistema consegue processar em tempo hábil, a amostragem reduz a taxa de chegada de dados através do descarte de tuplas. Um possível cenário de uso para a amostragem de dados é na execução de funções de junção (*join*) em banco de dados. Uma operação de junção com amostragem retorna um resultado aproximado, mas sem realizar o bloqueio da base de dados e mais rapidamente que a operação completa. O algoritmo clássico para manter uma amostragem aleatória em linha é a técnica *recevoir sampling*. O algoritmo mantém uma amostra de tamanho *s*, chamada reservatório. Quando novos fluxos de dados chegam, cada novo elemento tem uma probabilidade de substituir um elemento antigo do reservatório. Uma extensão desse algoritmo é manter uma amostra de tamanho *k* sobre uma janela deslizante de tamanho *n* dos dados mais recentes.

Descarte de carga (*Load Shedding*) refere-se ao processo de descarte de sequências de fluxos de dados quando a taxa de entrada excede a capacidade de processamento do sistema. Assim, o sistema é capaz de ter um comportamento adaptativo para atingir os requerimentos de latência. Tal técnica também recai sobre a perda de informações. Essa técnica é geralmente utilizada em sistemas de consultas dinâmicas. Na mineração de dados, o descarte de carga é difícil de ser usado, pois o descarte de blocos de dados do fluxo pode gerar a perda de dados úteis para a construção de modelos ou pode descartar padrões de interesse a serem identificados em uma série temporal.

Rascunho (*Sketching*) é processo de projetar um subconjunto aleatório de atributos, ou domínio, dos dados. A ideia chave é produzir resultados mais rápidos com limites de erro comprovados matematicamente. O rascunho faz uma amostragem vertical, excluindo colunas de atributos, dos dados que chegam em fluxo. O processo de rascunho é aplicado em diferentes fluxos de dados e em consultas de agregação [Gaber, 2012]. A principal desvantagem é a perda de acurácia já que os resultados são aproximados. Uma alternativa ao uso do rascunho em aprendizado de máquina em fluxos de dados é a análise de componentes principais (*Principal Components Analysis*- PCA), em que ao invés de se usar um subconjunto dos atributos, usa-se uma combinação linear dos atributos que maximize a variância dos dados.

Estruturas de dados de Sinopse (*Synopsis*) são estruturas de resumo de dados em memória. A ideia central é gerar uma aproximação do resultado, com uma complexidade de memória menor. A proposta, por exemplo, *hash sketches* visa a criação de um vetor de bits com tamanho L, em que $L = log(N)$, sendo N o número de dados. Seja $lsb(y)$ a função que denota a posição do bit 1 menos significativo na representação binária de y. Os dados x que chegam são mapeados em uma posição do vetor de bits através de uma função *hash(x)*, uniforme, e da função $lsb(hash(x0)$, que marca no vetor de bits a ocorrência do dado. A partir dessa proposta, pode-se definir que sendo R a posição do valor zero mais à direita no vetor de bits, é possível estimar o número de elementos no vetor de bits como $E[R] = log(\phi d)$, em que $\phi = 0.7735$ e, então, $d = 2^R/\phi$. O uso de histogramas para estimar a frequência relativa de amostras em fluxo também constitui estruturas de dados de sinopse.

Agregação é a técnica de resumir os dados que chegam. O resumo pode ser feito na forma de média, variância, máximo ou mínimo. O custo de memória ao usar essa técnica é muito baixo, mas a técnica falha caso o fluxo de dados tenha muita variação. Vale ressaltar como técnicas de agregação o cálculo recursivo da média, dado por

$$\overline{x_i} = \frac{(i-1) \times \overline{x_{i-1}} + x_i}{i}, \tag{9}$$

em que x_i é média atual, após i rodadas, e o cálculo recursivo da variância, dado por

$$\sigma_i = \sqrt{\left(\sum x_i^2 - \left(\sum x_i\right)^2 / i\right) / (i-1)}. \tag{10}$$

Wavelets é uma transformada em que o sinal passa a ser representado por uma soma de formas de onda, mais simples e definidas na construção da transformada, em diferentes escalas e posições. A transformada *wavelet* tem como objetivo capturar tendências em funções numéricas, decompondo o sinal em um conjunto de coeficientes. De forma semelhante ao que ocorre com a redução de características usando PCA, os coeficientes mais baixos podem ser descartados. O conjunto reduzido de coeficientes é usado em algoritmos que operam sobre o fluxo. A transformada mais usada no contexto de algoritmos de fluxo é a *Haar wavelet*.

As técnicas para processamento de dados em fluxo baseadas em tarefas são métodos que modificam técnicas existentes, ou criam novas, para atender ao desafio computacional do processamento em fluxo.

Algoritmos de aproximação retornam resultados aproximados da computação limitados por limiares de erro [Gaber, 2012]. Na mineração de dados em fluxos, em especial, os algoritmos com resultados aproximados têm notoriedade, pois os resultados são esperados de serem gerados de maneira contínua, rápida e com recursos computacionais limitados.

Janelas de tempo são comumente usadas para resolver consultas em fluxos de dados sem fim definido. Ao invés de executar uma computação sobre o dado completo, a computação é realizada sobre um subconjunto de dados, eventualmente mais de uma vez sobre os mesmos dados. Nesse modelo, uma marcação de tempo (*timestamp*) é associado a cada dado entrante. A marcação de tempo é usada para definir se um dado está dentro ou fora da janela considerada. A computação é executada somente sobre os dados que estão dentro da janela considerada. Algumas abordagens distintas de janelas são a janela de ponto de referência (*landmark window*), janela saltitante, a janela deslizante (*sliding window*) e a janela enviesada (*tilted window*).

A janela de ponto de referência identifica pontos relevantes nos fluxos de dados e, a partir de então, calcula os operadores de agregação a partir desse ponto de referência. Janelas sucessivas compartilham o mesmo início e são de tamanhos crescentes. Um ponto de referência, por exemplo, pode ser o início de um novo dia para uma agregação de dados diária. A janela saltitante é uma estrutura que considera um determinado número fixo de amostras e quando um conjunto subsequente com o número suficiente de amostras chega, as anteriores são descartadas e computação passa a ser feita sobre o novo conjunto de amostras. A janela saltitante usa o conjunto de tamanho fixo de amostras apenas uma vez

e cada amostra só é usada em um conjunto. A janela deslizante, por sua vez, é uma estrutura de dados de tamanho fixo que se insere amostras mais recentes e retira as amostras mais antigas de modo similar ao modelo de *primeiro a chegar, primeiro a sair*. Tal estrutura é interessante computacionalmente, pois na maioria dos casos o passado total não é tão relevante quanto o passado recente. Assim, a janela deslizante considera somente um número fixo de amostras no passado mais recente. A estrutura da janela deslizante é bastante utilizada para o cálculo de médias móveis. Por fim, a janela enviesada cria diferentes resoluções para a agregação dos dados. Diferente das janelas anteriores em que as amostras ou estavam dentro da janela ou fora, na janela enviesada, as amostras mais recentes são tratadas com granularidade fina, enquanto amostras do passado são agrupadas em granularidades distintas. Quanto mais no passado, mais grossa a granularidade de agrupamento das amostras.

4.5.2. Métodos de Aprendizado de Máquina *Online*

Uma primeira abordagem para o tratamento de grandes massas de dados em fluxo é o uso de métodos de aprendizado que sejam capazes de aprender com dados infinitos em tempo finito. A ideia central é aplicar métodos de aprendizado que limitem a perda de informação ao usar modelos com dados finitos em relação a modelos com dados infinitos. Para tanto, a perda de informação é medida em função do número de amostras usadas em cada etapa de aprendizado e, então, o número de amostras usadas em cada etapa é minimizado mantendo-se o limiar de perda conservado.

A resolução do problema de o quanto de informação se perde ao diminuir o número de amostras é dada usando o limite de Hoeffding (*Hoeffding bound*) [Gama et al., 2014]. Considerando uma variável aleatória real x, cujo valor está contido no intervalo R, supõe-se que são feitas n observações independentes da variável e computa-se a média $\bar{r}$. O *Hoeffding bound* garante que, com probabilidade $1-\delta$, a média verdadeira da variável é dada por, pelo menos, $\bar{x}-\varepsilon$, em que

$$\varepsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}. \tag{11}$$

O *Hoeffding bound* é independente da distribuição que gera a variável x. A partir desse resultado, desenvolvem-se os algoritmos de aprendizado de máquina para treinamento com fluxos de dados. Contudo, vale ressaltar que se assume que os valores gerados pela variável x advêm de um processo estocástico estacionário. Nos casos em que há uma mudança no processo que gera a variável usada no treinamento de métodos de aprendizado por fluxo é dito que ocorre uma mudança de conceito (*concept drift*) e, portanto, faz-se necessário um novo treinamento do método de aprendizado [Wang et al., 2013].

As abordagens típicas para aprender novas informações envolvem a manutenção do comportamento estocástico dos dados ou o descarte do classificador existente e, consequentemente, o retreinamento com os dados acumulados até o momento. As abordagens que consideram o fim da estabilidade estatística dos dados, o processo deixa de ser estocástico, resultam na perda de todas as informações adquiridas anteriormente, o que é conhecido como o esquecimento catastrófico. Dessa forma, Polikar *et al.* definem que algoritmos de aprendizado incremental devem satisfazer os requisitos de

obter informações adicionais de novos dados; de não exigir acesso aos dados originais, usados para treinar o classificador já existente; de preservar o conhecimento previamente adquirido, ou seja, não deve sofrer esquecimento catastrófico; e de ser capaz de acomodar novas classes que possam ser introduzidas por novos dados. Assim, classificadores que adotem o aprendizado incremental não requerem o treinamento de todo o classificador no caso de uma mudança no comportamento estacionário dos dados em fluxo.

Algoritmos em linha de árvores de decisão incrementais

Esses algoritmos são divididos em duas categorias: i) árvores construídas através de um algoritmo guloso de busca, em que a adição de novas informações envolvem a reestruturação completa da árvore de decisão e ii) árvores incrementais que mantêm um conjunto suficiente de estatísticas em cada nó da árvore para realizar um teste de divisão do nó, tornando a classificação mais específica, quando as estatísticas acumuladas no nó são favoráveis à divisão. Um exemplo desse tipo de árvore incremental é o sistema *Very Fast Decision Tree* (VFDT). O objetivo do sistema VFDT é projetar um método de aprendizado por árvore de decisão para conjuntos de dados extremamente grandes, potencialmente infinitos. A ideia central é que cada amostra de informação seja lida uma única vez e em um pequeno tempo de processamento. Isso possibilita o gerenciamento direto das fontes de dados em linha (*online*), sem armazenar as amostras. Para encontrar o melhor atributo que deve ser testado em um determinado nó, pode ser suficiente considerar apenas um pequeno subconjunto das amostras de treinamento que passam por esse nó. Assim, dado um fluxo de amostras, as primeiras serão usadas para escolher o teste da raiz; uma vez que o atributo da raiz é escolhido, as amostras subsequentes são transmitidas ao nós folhas correspondentes e usadas para escolher os atributos apropriados nesses nós, e assim por diante, recursivamente.

Em um sistema VFDT, a árvore decisão é aprendida recursivamente, substituindo folhas por nós de decisão. Cada folha armazena estatísticas suficientes sobre valores de atributos. Estatísticas suficientes são aquelas necessárias por uma função de avaliação heurística que avaliam o mérito de testes de divisão de nós baseados em valores de atributos. Quando uma amostra está disponível, ela atravessa a árvore da raiz até uma folha, avaliando o atributo apropriado em cada nó e seguindo o ramo correspondente ao valor do atributo na amostra. Quando a amostra chega à folha, as estatísticas são atualizadas. Então, as condições possíveis baseadas nos valores dos atributos são avaliadas. Caso haja suporte estatístico suficiente em favor de um teste de valor de um atributo em relação aos demais, a folha é convertida em nó de decisão. O novo nó de decisão vai ter tantos descendentes quantos valores possíveis para o atributo de decisão escolhido. Os nós de decisão mantêm somente as informações sobre o teste de divisão instalado no nó. O estado inicial da árvore consiste em uma folha única que é a raiz da árvore. A função de avaliação heurística é o Ganho de Informação (*Information Gain*), denotado por $H(\cdot)$. As estatísticas suficientes para estimar o mérito de um atributo nominal são os contadores n_{ijk} que representam o número de exemplos da classe k que chegam à folha, em que o atributo j recebe o valor i. O ganho de informação mede a quantidade de informação necessária para classificar uma amostra que chega ao nó: $H(A_j) = info(amostras) - info(A_j)$. A informação do atributo j é dada por

$$info(A_j) = \sum_i P_i \left(\sum_k -P_{ik} log_2(P_{ik}) \right), \tag{12}$$

em que $P_{ik} = \frac{n_{ijk}}{\sum_a n_{ajk}}$ é a probabilidade de se observar o valor de o atributo i dada a classe k e $P_i = \frac{\sum_a n_{ija}}{\sum_a \sum_b n_{ajb}}$ é a probabilidade de observar o valor do atributo i.

No sistema VFDT usa-se o limiar de Hoeffding, Equação 11, para decidir quantas amostras são necessárias observar antes de instalar um teste de separação em cada folha. Sendo $H(\cdot)$ a função de avaliação do atributo, para o ganho de informação, $H(\cdot)$ é o $log_2(||K||)$, em que K é o conjunto de classes. Seja x_a o atributo com maior valor de $H(\cdot)$, x_b o atributo com o segundo maior valor de $H(\cdot)$ e $\overline{\Delta H} = \overline{H}(x_a) - \overline{H}(x_b)$, a diferença entre os dois melhores atributos. Então, se $\overline{\Delta H} > \varepsilon$, com n amostras observadas na folha, o limiar de Hoeffding define com probabilidade $1-\delta$ que x_a é realmente o atributo com o maior valor na função de avaliação. Assim, a folha deve ser transformada em um nó de decisão que divide em x_a.

A avaliação da função para cada amostra pode ser muito custosa e, portanto, não é eficiente computar $H(\cdot)$ na chegada de cada nova amostra. A proposta VFDT só computa a função de avaliação de atributo quanto um número mínimo de amostras, definido pelo usuário, são observadas desde a última avaliação. Quando dois ou mais atributos têm os mesmos valores de $H(\cdot)$ continuamente, mesmo com um grande número de amostras, o limiar de Hoeffding não é capaz de decidir entre eles. Então, o VFDT introduz uma constante τ em que se $\overline{\Delta H} < \varepsilon < \tau$, então a folha é convertida em um nó de decisão e o teste de decisão é baseado no melhor atributo. Gama *et al.* generalizam o funcionamento do sistema VFDT para atributos numéricos [Gama et al., 2014].

Naive Bayes Incremental

Dado um conjunto de treinamento $\chi = (\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_n)$, em que $\mathbf{x} \in \mathbb{R}^D$ são amostras com $D-dimensionais$ no espaço de atributos e $y \in 1, ..., K$ são as classes correspondentes para um problema de classificação em K classes, formula-se o Teorema de Bayes como

$$p(y = i|x) = \frac{p(i)p(\mathbf{x}|i)}{p(\mathbf{x})}, \tag{13}$$

em que $p(i)$ é a probabilidade *a priori* de ocorrência de uma amostra da classe e $p(y|\mathbf{x})$ é a distribuição de probabilidades desconhecida do espaço de atributos $\mathbf{x}$ e marcada com a classe i. Uma estimativa para a distribuição desconhecida é assumir a independência dos atributos dada a marcação da classe, levando a

$$p(x_1, x_2, ..., x_D|i)\ p(x_1|i)p(x_2|i)...p(x_D|i), \tag{14}$$

em que x_d representa a d-ésima dimensão no vetor de atributos $\mathbf{x}$. Assim, o classificador bayesiano é descrito como

$$F(\mathbf{x}) = arg\ \max_{i} \prod_{d=1}^{D} p(x_d|i). \quad (15)$$

Assim, a classificação é calculada fazendo a multiplicação de todas as probabilidades das classes para o valor dos atributos da amostra [Godec et al., 2010].

A versão incremental do classificador bayesiano prevê a atualização dos valores das probabilidades das classes por atributos conforme novas amostras são processadas. Uma abordagem para permitir o armazenamento eficiente das funções de probabilidade conforme as amostras chegam é realizar a discretização e armazenar histogramas dos atributos. A proposta IFFD (*incremental flexible frequency discretization*) apresenta um método para discretização de atributos quantitativos em uma sequência de intervalos de tamanhos flexíveis. Essa abordagem permite a inserção e a divisão de intervalos.

Aprendizado Incremental por Agregados de Classificadores

O algoritmo ARTMAP baseia-se na geração de novos agrupamentos de decisão em resposta a novos padrões que são suficientemente diferentes de instâncias vistas anteriormente. O valor de quanto diferente é um padrão já conhecido de um novo é controlado por um parâmetro de vigilância definido pelo usuário. Cada agrupamento aprende em um hiper-retângulo que é uma porção diferente do espaço de características, em um modo não supervisionado, que são então mapeados para classes alvo. Agrupamentos são sempre mantidos, o ARTMAP não sofre o esquecimento catastrófico. Além disso, ARTMAP não requer acesso a dados previamente vistos e pode acomodar novas classes. Contudo, o ARTMAP é muito sensível à seleção do parâmetro de vigilância, aos níveis de ruído nos dados de treinamento e à ordem em que os dados de treinamento chegam.

O algoritmo AdaBoost (*adaptive boosting*) gera um conjunto de hipóteses e as combina através da votação da maioria ponderada das classes previstas pelas hipóteses individuais. As hipóteses são geradas pelo treinamento de um classificador fraco[3], usando instâncias extraídas de uma distribuição atualizada periodicamente dos dados de treinamento. Esta atualização de distribuição garante que instâncias mal classificadas pelo classificador anterior sejam mais provavelmente incluídas nos dados de treinamento do próximo classificador. Assim, os dados de treinamento de classificadores consecutivos são voltados para instâncias cada vez mais difíceis de classificar.

O algoritmo de aprendizagem incremental Learn ++ é inspirado pelo AdaBoost, originalmente desenvolvido para melhorar o desempenho de classificação de classificadores fracos. Em essência, Learn ++ gera um conjunto de classificadores fracos, cada um treinado usando uma distribuição diferente de amostras de treinamento. As saídas desses classificadores são então combinadas usando o regime de votação por maioria para obter a regra final de classificação. O uso de classificadores fracos é interessante, pois a instabilidade para que construam suas decisões é suficiente para que cada decisão seja diferente das demais, para que pequenas modificações em seus conjuntos de dados de treinamento sejam refletidas em classificações distintas.

[3] Algoritmos de classificação que a acurácia é próxima à classificação aleatória.

K-Médias Incremental

O algoritmo de agrupamento k-médias executa um critério de otimização iterativo da soma das distâncias ao quadrado de todos os pontos ao centro de cada agrupamento. No caso de um problema de agrupamentos por aprendizado de máquina não-supervisionado, define-se que N é o conjunto de entradas $\overline{x}_1, \overline{x}_2, ..., \overline{x}_N$, em que $\overline{x}_i \in R^n$. O problema consiste, então, em encontrar $K << N$ agrupamentos $c_1, c_2, ..., c_k$ com centros em $\overline{\omega}_1, \overline{\omega}_2, ..., \overline{\omega}_k$, respectivamente, tal que a distância D da soma ao quadrado das distâncias dos dados aos centros dos agrupamentos a que pertencem seja mínima. A distância D é equivalente ao erro médio quadrático (*Mean Sqaure Error* – MSE) e é dada por

$$D = \frac{1}{N} \sum_{j=1}^{K} \sum_{\overline{x} \in c_j} (\overline{x} - \overline{\omega}_j)^2. \tag{16}$$

As iterações do algoritmo clássico consistem em uma inicialização, em que são escolhidos K centros, e os elementos são classificados pela regra do vizinho mais próximo. Após, os centros dos agrupamentos são atualizados através do cálculo do centroide de cada agrupamento. Nas iterações seguintes, os dados são reclassificados de acordo com os novos centroides calculados. As iterações são executadas até a convergência do algoritmo quando os centros dos agrupamentos calculados na iteração i são idênticos aos de $i+1$.

A variação do algoritmo das k-médias para o tratamento sequencial dos dados consiste em recalcular os centros dos agrupamentos sempre que uma nova amostra chega. Essa variação depende de todos os dados já processados serem acessados novamente para o recálculo dos centros dos agrupamentos, o que gera uma demanda de recursos computacionais substancialmente alta, inviabilizando o seu uso. A variação do algoritmo usando blocos sequenciais, por sua vez, viabiliza a utilização do algoritmo de k-médias *online*, pois o agrupamento é realizado sobre blocos de dados acumulados. Cada bloco é usado por l épocas do algoritmo de k-médias e os resultados dos centros dos agrupamentos do bloco i são usados como os centros iniciais da iteração sobre o bloco $i+1$. A variação incremental do algoritmo define que o bloco é usado somente uma vez [Aaron et al., 2014]. A validade do resultado do algoritmo incremental reside no fato de que a distribuição de probabilidades dos atributos das amostras não muda, ou muda lentamente, entre blocos.

4.5.3. Aprendizado por Reforço

O aprendizado por reforço (*Reinforcement Learning* - RL) baseia-se em um agente interagindo com o ambiente, aprendendo uma política ótima, por tentativa e erro, para problemas de tomada de decisão sequenciais em campos de ciências naturais e sociais e engenharia [Li, 2018]. O agente de aprendizado por reforço interage com o ambiente através do tempo. A cada passo t de tempo, o agente aplica uma política $\pi(a_t|s_t)$, em que s_t é o estado que o agente recebe de um espaço de estados S e a_t é uma ação selecionada pelo agente em um espaço de ações A. Dessa forma, o agente mapeia o estado s_t em uma ação a_t para receber uma recompensa, ou penalidade, escalar r_t e realizar a transição para o próximo estado s_{t+1}. A transição ocorre de acordo com a dinâmica ou o modelo do ambiente. A função $R(s,a)$ modela a recompensa do agente e a função $P(s_{t+1}|s_t, a_t)$ modela a probabilidade de transição de estados do agente. O agente continua a execução do ciclo até atingir um estado terminal, quando o ciclo é reinicado. O retorno $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ é

o acumulo de recompensa descontada por um fator $\gamma \in [0,1)$. A ideia central do aprendizado por reforço é que o agente maximize a expectativa de um retorno de longo prazo para cada estado. Vale ressaltar que os mecanismos de aprendizado por reforço assumem que o problema satisfaz a propriedade de Markov, em que o futuro depende apenas do estado atual e da ação e não do passado. Assim, o problema é formulado como um Processo de Decisão de Markov (*Markov Decision Process* - MDP), definido pela 5-tupla (S,A,P,R,γ).

A aprendizagem de diferença temporal é um dos pilares do aprendizado por reforço, já que a aprendizagem de diferença temporal se refere aos métodos de aprendizagem para avaliação da função de valor, SARSA e *Q-learning*. O aprendizado de diferença temporal aprende a função de valor $V(s)$ diretamente da experiência com o erro da diferença temporal, com inicialização independente do modelo, *online* e totalmente incremental. O aprendizado de diferença temporal é um problema de previsão. A regra de atualização é $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$, em que α é a taxa de aprendizagem e $r + \gamma V(s') - V(s)$ é o chamado erro de diferença temporal. O uso do algoritmo do gradiente descendente generalizado garante a convergência dessa gama de problemas [Lobato et al., 2018].

O problema de previsão ou avaliação de política com aprendizado por reforço consiste em calcular o estado ou função de valor de ação para uma política. O problema de controle é encontrar a política ótima. O algoritmo SARSA avalia a política com base em amostras da mesma política e refina a política através de uma metodologia gulosa em relação aos valores de ação. Em métodos fora da política (*off-policy*), o agente aprende uma função de valor ou política ideal, talvez seguindo uma política comportamental não relacionada. O algoritmo *Q-learning*, por exemplo, tenta encontrar diretamente valores de ação para a política ótima, não necessariamente se ajustando à política que gera os dados. Assim, a política encontrada pelo *Q-learning* é geralmente diferente da política que gera as amostras [Li, 2018]. As noções de métodos na política e fora da política relacionam-se com as ideias de avaliar as soluções encontradas com as mesmas políticas que geram os dados ou avaliar com políticas ligeiramente diferentes. Avaliar com políticas diferentes refere-se a usar um modelo para gerar os dados, enquanto em métodos sem modelo o agente aprende através de tentativas e erro nas ações tomadas. Os algoritmos de aprendizado por reforço podem executar em modo *online* ou *offline*. No modo online, os algoritmos de treinamento são executados em dados adquiridos em fluxos sequenciais. No modo offline ou em lote, os modelos são treinados em todo o conjunto de dados.

O aprendizado por reforço é uma estratégia presente em trabalhos que realizam a extração de conhecimento e a tomada de decisões em redes sem fio [Liu e Yoo, 2017, Tabrizi et al., 2011, Chen e Qiu, 2011]. Liu e Yoo propõe um esquema que usa o algoritmo Q-learning para alocar dinamicamente sub-quadros em branco para que ambos LTE-U e Sistema WiFi pode coexistir com sucesso [Liu e Yoo, 2017]. A técnica de ajuste é baseada no aprendizado por reforço. A proposta introduz uma nova estrutura de quadro LTE-U, que além de alocar subquadros em branco, também reduz o atraso de LTE. Na proposta as ações do algoritmo são modeladas através de uma tupla que contém o número total subquadros em um bloco de quadros e a porção de subquadros para LTE-U. Os estados são modelados como o conjunto de todas as ações tomadas até aquele estado. Tabrizi *et al.* advogam que a coexistência de várias tecnologias sem fio, as comunicações de próxima

geração tendem a usar um sistema integrado de redes, em que pontos de acesso WiFi e estações base da rede de celular trabalham em conjunto para maximizar a Qualidade de Serviço (QoS) do usuário móvel [Tabrizi et al., 2011]. Dessa forma, os autores propõem que os dispositivos móveis com diferentes tecnologias de acesso selecionem redes e alternar facilmente de um ponto de acesso ou estação base para outro para melhorar o desempenho do usuário. A proposta baseia-se na seleção de rede com o objetivo de maximizar a QoS, formulada como um Processo de Decisão de Markov e, portanto, um algoritmo baseado no aprendizado por reforço obtido para selecionar a melhor rede baseada na carga de rede atual e também nos possíveis futuros estados de rede. Chen e Qiu propõem uma abordagem para o sensoriamento cooperativo de espectro para rádios definidos por software baseado no algoritmo *Q-learning* [Chen e Qiu, 2011]. No sensoriamento do espectro cooperativo usando Q-learning, o conjunto de estados é composto por todas as combinações de um vetor de bits em que cada usuário marca um bit e o conjunto de ações é $0, 1$, em que "0" significa que o canal está no estado "disponível" para usuários secundários e uma ação com índice "1" significa que o canal está em "ocupado" e indisponível usuários secundários. Atribuir valores para recompensa r para Q-learning pode ser complicado. No algoritmo proposto, as recompensas são positivas quando a ação está de acordo com a ocupação do canal e negativas em caso contrário.

4.5.4. Aprendizado Profundo - *Deep Learning*

O aprendizado profundo (*deep learning*) consistem em uma classe de técnicas de aprendizado de máquina que exploram várias camadas de processamento de informações não lineares para extração e transformação de recursos, usando aprendizado supervisionados ou não supervisionados, e para análise e classificação de padrões [Deng e Yu, 2014]. Assim, o aprendizado profundo baseia-se em aprender vários níveis de representação e abstração que ajudam a compor o sentido dos dados. Em especial, essa classe de técnicas está na interseção de áreas de pesquisas de redes neurais, inteligência artificial, modelagem gráfica, otimização, reconhecimento de padrões e processamento de sinais e, portanto, são geralmente usadas no processamento de imagens, som e texto [Kwon et al., 2017]. Os principais aspectos das técnicas que se caracterizam por serem de aprendizado profundo são os modelos que consistem de múltiplas camadas ou estágios de processamento não-lineares e os métodos para aprendizagem supervisionada ou não supervisionada de representação de características em camadas sucessivamente mais altas e mais abstratas. Dessa forma, a ideia chave do aprendizado profundo é gerar novas representações dos dados a cada camada, aumentando o grau de abstração da representação. A popularidade crescente das técnicas de aprendizado profundo deve-se ao aumento acelerado da capacidade de processamento de *chips*, como os das unidades gráficas (GPUs); ao aumento significativo dos dados disponibilizados para o treinamento dos modelos; e aos avanços nas pesquisas de aprendizado de máquina [Kwon et al., 2017], que permitiram a exploração de funções complexas não-lineares de composição, a aprendizagem distribuída e hierárquica e uso efetivo de dados rotulados e não rotulados.

As arquiteturas e técnicas de aprendizado profundo podem ter uso voltado para a síntese/geração ou reconhecimento/classificação e, assim, são geralmente classificadas [Deng, 2014]. **Arquiteturas profundas geradoras** caracterizam as propriedades de correlação de alta ordem dos dados observados para a análise ou síntese de padrões e/ou

caracterização das distribuições estatísticas conjuntas dos dados observados e de suas classes associadas. **Arquiteturas profundas discriminativa** proveem valores para realizar a discriminação em classes de padrões e, por vezes, caracterizando as distribuições *a posteriores* de classes condicionadas aos dados observados. **Arquiteturas híbridas profundas** têm por objetivo discriminar dados em classes, mas é assistida com os resultados de arquiteturas geradoras através da otimização e/ou regularização, ou quando critérios discriminativos são usados para aprender os parâmetros em modelos generativos. As arquiteturas geradoras estão associadas à identificação e ao reconhecimento de padrões ocultos nos dados observados, enquanto as arquiteturas discriminativas estão associadas à classificação de dados observados em classes definidas. Ressalta-se que as arquiteturas geradoras relacionam-se a problemas de aprendizado não-supervisionado, enquanto as arquiteturas discriminativas, aos problemas de aprendizado supervisionado. No processo de aprendizado não supervisionado não existem dados rotulados e, assim, principal objetivo desse processo é gerar dados rotulados aplicando algoritmos de aprendizado não supervisionados, como Máquina Restrita de Boltzmann (*Restricted Boltzmann Machine* - RBM), rede de crença profunda (*Deep Belief Network* - DBN), rede neural profunda (deep neural network - DNN), *AutoEncoders* generalizados, rede neural recorrente (*recurrent neural network* - RNN) [Kwon et al., 2017].

As Máquinas Retritas de Boltzmann são modelos generativos probabilísticos capazes de extrair automaticamente características dos dados de entrada usando um algoritmo de aprendizado completamente não supervisionado [Kim et al., 2010]. As RBMs consistem em uma camada oculta e uma camada de neurônios visíveis com conexões entre os neurônios ocultos e visíveis representados por uma matriz de pesos. Para treinar uma RBM, as amostras de um conjunto de treinamento são usadas como entrada para a RBM através dos neurônios visíveis e a rede gera amostras alternadamente para trás e para frente entre os neurônios visíveis e ocultos. O objetivo do treinamento é aprender pesos das conexão entre neurônios visíveis e ocultos e aprender os vieses de ativação de neurônios, de modo que o RBM aprenda a reconstruir os dados de entrada durante a fase em que os neurônios visíveis dos neurônios ocultos são amostrados. Cada processo de amostragem é essencialmente uma multiplicação de matrizes entre um lote de amostras de treinamento e a matriz de peso, seguido por uma função de ativação de neurônio, que em muitos casos é uma função sigmóide $(1/(1+e^{-x}))$. A amostragem entre as camadas oculta e visível é seguida por uma ligeira modificação nos parâmetros, definida pela taxa de aprendizagem α, e repetida para cada lote de dados no conjunto de treinamento, e por tantas épocas quantas forem necessárias para alcançar a convergência [Kim et al., 2010].

A rede de crença profunda (DBN) consiste em múltiplas camadas de variáveis estocásticas e ocultas e relaciona-se com a RBM, pois consiste na composição e o empilhamento de várias RBMs. A composição de várias RBM permite que muitas camadas ocultas treinem dados de forma eficiente por meio de ativações de um RBM para estágios adicionais de treinamento [Kwon et al., 2017, Kim et al., 2010].

As Redes Neurais de Convolução (CNNs) representam uma das formas mais comuns de Redes Neurais Profundas (DNN) [Sze et al., 2017]. As CNNs são compostas por múltiplas camadas convolucionais e cada camada gera um mapa de características, uma abstração de nível mais alto dos dados de entrada, que preserva informações essenciais e únicas. Cada uma das camadas convolucionais na CNN é composta principalmente por

convoluções de alta dimensão, em que as ativações de entrada de uma camada são estruturadas como um conjunto de mapas de características de entrada, chamado de canal. Por esse motivo, as CNNs são geralemente usadas em aplicações de processamento de sinais. Cada canal é convolvido com um filtro distinto da pilha de filtros. O resultado dessa computação são as ativações de saída que compõem um canal de mapa de características de saída. Por fim, vários mapas de características de entrada podem ser processados juntos em lote para melhorar potencialmente a reutilização dos pesos do filtro.

AutoEncoder tem sido tradicionalmente usado para redução de dimensionalidade e aprendizado de características. A ideia fundamental do *AutoEncoder* a presença de uma camada oculta h que se refere à entrada e outras duas partes principais, a função de codificação $h = f(x)$ e decodificação ou reconstrução $r = g(h)$. O principal objetivo deste conceito é que tanto o codificador quanto o decodificador sejam treinados juntos e a discrepância entre os dados originais e sua reconstrução possa ser minimizada. O *AutoEncoder* profundo é uma parte de um modelo não supervisionado [Kwon et al., 2017].

Redes neurais convencionais baseiam-se no princípio de que todos os pontos de dados são independentes. Por esse motivo, se os pontos de dados estiverem relacionados a tempo ou espaço, a chance de perder o estado da rede é alta. Já as redes neurais recursivas (RNN) são baseadas em sequências, de modo que elas podem modelar uma entrada ou saída que seja composta de elementos independentes em sequência. A RNN pode ser usada em aprendizado não supervisionado ou supervisionado. Quando usada em aprendizado não supervisionado, a predição da sequência de dados da amostras de dados anteriores é possível, mas apresenta dificuldade de treinamento [Kwon et al., 2017].

O aprendizado profundo é normalmente treinado com a abordagem de gradiente estocástico descendente [Lobato et al., 2018], em que um exemplo de treinamento com o rótulo conhecido por vez é usado para atualizar os parâmetros do modelo. A estratégia pode ser usada para o aprendizado em fluxo (*online*). Contudo, uma estratégia para acelerar a aprendizagem consiste em realizar as atualizações em minilotes de dados (*mini-batches*) em vez de proceder sequencialmente com uma amostra de cada vez [Chen e Lin, 2014]. As amostras em cada minilote são o mais independentes possível e essa abordagem fornece um bom equilíbrio entre o consumo de memória e o tempo de execução.

Trabalhos recentes usam o aprendizado profundo para inferir características e comportamentos de redes sem fio. A proposta BiLoc desenvolve um algoritmo baseado em aprendizado profundo para explorar dados bi-modais, ângulo de fase estimado de chegada e amplitudes médias na interface de rádio de 5 GHz para redes sem fio, para gerar impressões digitais de localizações internas (*indoor*) de dispositivos [Wang et al., 2017]. O aprendizado profundo produz impressões digitais baseadas em características a partir dos dados bi-modais no estágio de treinamento off-line. Os pesos na rede de *autoencoder* profundo são as impressões digitais baseadas em características para cada posição. Wang *et al.* comparam duas abordagens de localização interna usando aprendizado profundo, uma com *autoencoders* e outra com redes neurais convulucionais [Wang et al., 2018]. Os autores concluem que a abordagem baseada em *autoencoders* apresentam menor erro na inferência da posição interna. Turgut *et al.*, por sua vez, usam *autoencoder* profundo para realizar a localização interna de dispositivos sem fim, porém consideram como características inicias a potência de sinal recebido de 26 pontos de acesso [Turgut et al.,

2019]. Wang *et al.* usam o aprendizado profundo para o reconhecimento de atividade mais preciso e robusto em canais de redes sem fio. A ideia central é selecionar ativamente canais WiFi disponíveis com boa qualidade e alternar entre canais adjacentes para formar um canal estendido. Wang *el al.* buscam por padrões sequenciais de uso de canal, então adotam um modelo de uma rede neural recursiva [Wang et al., 2018].

4.6. Discussão, Tendências e Desafios de Pesquisa

As principais aplicações baseadas em atuação automática para monitoramento de redes sem fio consistem em detecção de anomalia, geração de perfis de usuários, aplicações, estações e rede, abordagens baseadas em comportamento, visualização e otimização [Li et al., 2013]. Li *et al.* apontam que as principais aplicações de processamento de dados em fluxo de redes são o monitoramento de aplicações, estações e rede; a classificação do comportamento de aplicações, a segurança e a detecção de intrusão em redes. O monitoramento de rede fornece informações sobre os elementos de rede e a visão consolidada da rede é usada detectar problemas e propor soluções eficientes. O monitoramento de aplicações fornece informações sobre o uso na rede permite planejamento e alocação de recursos. O monitoramento de estações fornece informações sobre o padrão de utilização do usuário, da rede e da aplicação e é usado para planejamento da rede, do controle de acesso e para a verificação de violações de políticas de segurança. A classificação de do comportamento de aplicações permite identificar qual o tipo da aplicação em execução, mesmo quando o tráfego é criptografado. Outra a aplicação de dados em fluxo é a inferência da identidade de usuários baseada somente no comportamento dos fluxos de comunicação na rede. Por fim, Li *et al.* ainda ressaltam o uso do processamento de dados em fluxo como uma componente importante para prover segurança em rede através da identificação de anomalias ou da classificação de tráfego em busca de assinaturas de *worms*, varredura de portas, execução de *botnets*, ataques de negação de serviço, como também para a validação de novas políticas de rede.

Há ainda desafios de pesquisa para o tratamento de grandes massas de dados em tempo real. Os principais desafios do processamento em fluxo de dados para redes sem fio relacionam-se com as cinco dimensões principais no processamento de grandes massas de dados, volume, velocidade, variedade, veracidade e valor [Chen e Lin, 2014].

Estimação de erros nos dados relaciona-se com a granularidade das medições de realizadas por ferramentas de montiramento de redes, tais como SNMP, Netflow e OpenFlow. Erros podem acontecer nos processos de amostragem, transporte ou coleta e, portanto, associam-se à veracidade dos dados. Métodos para a estimação de intervalos de confiança, boas práticas na amostragem e a caracterização de erros mais comuns têm sido estudados e propostos para mitigar e efeito dos erros dos dados amostrados nas ações de controle e gerenciamento da rede [Li et al., 2013].

Significado dos dados relaciona-se ao valor dos dados e ao fato de que a maioria dos dados está dispersa em diferentes fontes e, portanto, o significado de dados coletados das várias fontes pode ser ligeiramente distinto. Isso pode afetar significativamente a qualidade dos resultados do aprendizado de máquina. Ontologia, web semântica e outras técnicas são propostas para mitigar problemas com o significado dos dados. Com base na modelagem de ontologias e na derivação semântica, padrões ou regras valiosos podem

ser descobertos, mas as técnicas de ontologias e web semântica ainda não estão maduras o suficiente.

Treinamento para o reconhecimento de padrões consiste em usar padrões rotulados para treinar os algoritmos de aprendizagem. Contudo, obter os rótulos implica custos de tempo e de computação, particularmente para os dados em fluxo em larga escala, sendo impactado pelo volume e pela velocidade em que os dados são criados. Em alguns casos, obter os padrões pode ser intratável computacionalmente. Outro desafio relacionado ao treinamento é o equilíbrio entre custo e precisão, chamado sobreajuste (*overfitting*), que é outro problema crítico em aberto.

Técnicas de integração de dados consistem na agregação de técnicas de mineração de dados, descoberta de conhecimento, computação em nuvem e aprendizado de máquina. Relaciona-se com a variedade dos dados, com a velocidade que novos dados são criados e com o volume de dados que chegam em fluxo. Contudo, cada técnica tem vantagens e desvantagens. Assim, uma tendência de pesquisa é a adoção de abordagens híbridas compostas de diversas técnicas.

Conjuntos de dados padrão e ambientes para pesquisa são é essenciais para identificar alguns problemas comuns em redes sem fio, juntamente com dados rotulados ou não rotulados para abordagens baseadas em aprendizado supervisionado ou não supervisionado. Para abordagens baseadas em aprendizado de reforço, devem ser construídos problemas de controle de rede padrão em conjunto com ambientes bem definidos a fim de permitir a comparação entre proposta de pesquisas [Sun et al., 2018].

Segurança e privacidade dos dados são desafios atuais para o processamento de grandes massas de dados, pois com o uso de tecnologias de mineração de dados e de aprendizado de máquina para analisar informações pessoais, é possível produzir resultados demasiadamente relevantes e interconectados que prejudicam a privacidade dos indivíduos. Informações pessoais que não são fornecidas *a priori* são inferidas através da correlação de dados provenientes de fontes distintas. Um dos principais desafios do processamento de grandes massas de dados é definir métodos eficientes e eficazes que gerem conhecimento preciso, preservando o desempenho da mineração e do aprendizado, ao passo que garantem a proteção das informações pessoais sensíveis;

Realização e aplicações consistem no uso do conhecimento gerado através do processamento em fluxo de grandes massas de dados em aplicações de tempo real. Em redes sem fio de grande porte, essas aplicações são temas de pesquisas recentes e relacionam-se com o controle da rede através da escolha de canal, mudança na associação entre clientes e pontos de acesso e na geração de perfis de uso da rede.

Outros desafios ainda abertos são identificados por Sun *et al*. Para a implementação de técnicas de aprendizado de máquina supervisionadas e não supervisionadas nas redes sem-fio, é essencial a criação de conjuntos de dados rotulados/não rotulados. Para abordagens baseadas em aprendizado de reforço, devem ser construídos problemas de controle de rede em ambientes bem definidos [Sun et al., 2018]. A transferência de aprendizado promete transferir o conhecimento aprendido de uma tarefa para outra tarefa semelhante. Evitando treinar modelos de aprendizado a partir do zero, o processo de aprendizado em novos ambientes pode ser acelerado, e o algoritmo aprendizado de

máquina pode ter um bom desempenho, mesmo com uma pequena quantidade de dados de treinamento. Portanto, a transferência de aprendizado é fundamental para a implementação prática de modelos de aprendizagem considerando o custo para treinamento sem conhecimento prévio. Usando o aprendizado de transferência, os operadores de rede podem resolver problemas novos, mas semelhantes, de maneira econômica.

Por outro lado, no gerenciamento da rede, o controle de *backhaul/fronthaul* heterogêneos baseado em aprendizado de máquina podem ser aplicados. Em futuras redes sem fio, várias soluções de *backhaul/fronthaul* coexistirão, incluindo fibra e cabo assim como sem-fio como a banda sub-6 GHz [Sun et al., 2018]. Cada solução consome quantidades de energia e largura de banda diferente. Portanto, para um bom desempenho do sistema, as técnicas baseadas em aprendizado de máquina podem ser usadas para selecionar soluções de *backhaul/fronthaul* adequadas com base nos padrões de tráfego extraídos e nos requisitos de desempenho dos usuários. Ainda no gerenciamento da rede, futuras atualizações das infraestruturas de rede sem fio devem ser desenvolvidas baseadas em técnicas de otimização amparadas por aprendizado de máquina.

Em outra vertente, o fatiamento da rede (*network slicing*) é defendido como uma maneira econômica de suportar diversos casos de uso. A ideia principal do fatiamento da rede é alocar recursos apropriados, incluindo recursos de computação, armazenamento em cache, *backhaul/fronthaul* e rádio sob demanda, para garantir os requisitos de desempenho de diferentes fatias isoladas. O fatiamento da rede pode se beneficiar do aprendizado de máquina quanto o mapeamento das demandas de serviço sobre os planos de alocação de recursos e, também, ao empregar transferência de aprendizado, o conhecimento sobre planos de alocação de recursos para diferentes casos de uso em um ambiente pode agir como conhecimento útil em outro ambiente, acelerando o processo de aprendizagem.

Projetos de pesquisa focam tanto no processamento em fluxo de grandes massas de dados, como no desenvolvimento de redes de experimentação de larga escala baseadas na tecnologia de redes sem fio. Alguns dos projetos de pesquisa focados em criar redes de experimentação sem fio abordados incluem o ORBIT, o NITOS, o OneLab, o FIT, o FUTEBOL e o FIBRE. Essas infraestruturas de testes permitem a experimentação em redes federadas incluindo diferentes tipos de dispositivos sem fio. Suas vantagens e desvantagens para aplicações de processamento em fluxo são avaliadas e discutidas.

OneLab[4] é uma iniciativa europeia que propõe a federação de ambientes de experimentação com diferentes tecnologias de acesso. O OneLab agrega ambientes de experimentação voltado para Internet das Coisas (IoTLab), um ambiente de experimentação de redes sem fio com nós Wi-Fi a/b/g/n (NITOS) e o ambiente PlanetLab europeu (PLE). **FIT**[5] é uma infraestrutura de testes aberta em larga escala para sistemas e aplicativos em comunicações sem fio e sensores. A FIT oferece uma grande variedade de tecnologias, tais como Internet das Coisas, redes sem fio e computação em nuvem, e também uma interface única de acesso ao sistema e um grande número de ferramentas de configuração e monitoramento. **FUTEBOL**[6] é um projeto de cooperação entre o Brasil e a Europa para desenvolver e implantar uma infraestrutura de experimentação que possibilite pesquisas

[4]Disponível em https://onelab.eu/.

[5]Disponível em https://fit-equipex.fr/.

[6]Disponível em http://www.ict-futebol.org.br/.

sobre o ponto de convergência entre redes ópticas e sem fio. **FIBRE**[7] é uma infraestrutura de experimentação que funciona como um laboratório virtual de larga escala para novas aplicações e modelos de arquitetura de rede. Organiza-se como uma federação de 11 ilhas de experimentação locais, entre as quais há ilhas de experimentação de redes sem fio, como, por exemplo, a disponível na Universidade Federal Fluminense (UFF). **PrEstoCloud**[8] foca em desenvolver pesquisas nas tecnologias de computação em nuvem e análise de dados em tempo real para fornecer uma arquitetura dinâmica e distribuída para gerenciamento proativo de recursos em nuvem, visando alocar o processamento na borda extrema da rede para reduzir a latência no processamento. PrEstoCloud combina a pesquisa de Big Data, computação em nuvem e computação em nuvem em tempo real. **Metron**[9] é um arcabouço de análise de segurança baseado no processamento de grandes massas de dados. A ideia chave desse arcabouço é permitir a correlação de eventos de segurança originados de fontes distintas e, para tanto, o arcabouço emprega como fonte de dados sensores na rede, registros de ações (*logs*) de elementos ativos de segurança em rede e fontes de telemetria. **Hogzilla**[10] é um Sistema de Detecção de Intrusão (IDS) de código aberto suportado por *softwares* livres, que fornecem detecção de anomalias na rede através do processamento em lote de grandes massas de dados. O Hogzilla emprega também ferramentas que permitem a visibilidade da rede.

4.7. Considerações Finais

As redes de acesso sem fio estão presentes em todos os ambientes. Os dados de acesso nessas redes também estão crescendo em volume, variedade e velocidade. Paralelamente, diversas pesquisas mostram que os dados dessas redes também apresentam grande valor, mas dependem da validação de sua veracidade. O valor dos dados de acesso em redes sem fio é crítico para o monitoramento, gerenciamento e controle da rede, mas também permite inferir conhecimentos sobre a infraestrutura, padrões de mobilidade, preferências e qualidade de experiência e dos serviços dos usuários. Assim, a análise dos dados gerados em uma rede de acesso sem fio de grande escala consiste em um problema de processamento grandes massas de dados. O paradigma de processamento de grandes massas de dados requer armazenamento, processamento e proteção eficientes em termos de latência de processamento, espaço de armazenamento em memória e banda de transmissão na rede. Esses requisitos representam também os principais desafios do processamento das grandes massas de dados em redes de acesso sem fio. Ademais, por se tratarem de fontes de dados ininterruptas que geram dados potencialmente infinitos e com atributos com grande variação, torna-se necessário a aplicação de ferramentas de processamento de grandes massas de dados em fluxo. Esse capítulo apresentou as principais ferramentas para o monitoramento de redes sem fio e aplicações de analíticas para redes. Destaca-se que metodologia como o aprendizado de máquina têm sido uma abordagem importante aplicada na classificação, na descoberta de padrões e na segurança das redes.

O capítulo discutiu os conceitos relacionados ao processamento em fluxo de dados e apresentou algoritmos de aprendizado de máquina incremental. Os algoritmos de

[7]Disponível em https://fibre.org.br/.

[8]Disponível em http://prestocloud-project.eu/.

[9]Disponível em http://metron.apache.org/.

[10]Disponivel em http://ids-hogzilla.org/.

aprendizado incremental tendem a assumir que a distribuição de probabilidade dos atributos considerados na formação do modelo não varia ao longo do tempo. Contudo, na ocorrência de uma variação, seja brusca ou amortecida no tempo, os algoritmos de aprendizado incremental tendem a falhar na extração de conhecimento dos dados entrantes. Um dos principais desafios para o processamento em fluxo de dados, então, é a detecção de mudanças nas estatísticas dos atributos dos dados entrantes e o desenvolvimento de algoritmos de aprendizado de máquina capazes de reagir a essas mudanças.

O capítulo analisou ainda as aplicações de última geração de aprendizado de máquina em redes sem fio e identificou as técnicas de aprendizado por agregado de classificadores, aprendizado profundo e aprendizado por reforço como promissoras para as próximas gerações de aplicações na extração do conhecimento no monitoramento de redes. Há ainda tendências de novas aplicações que adotam o aprendizado por transferência de conhecimento, em que o conhecimento extraído de um contexto é transferido para outra aplicação em execução em outro contexto. Dessa forma, a transferência de aprendizado evita treinar modelos de aprendizado a partir do zero e o processo de aprendizado em novos ambientes pode ser acelerado de modo em que o algoritmo de aprendizado de máquina pode ter um bom desempenho, mesmo com uma pequena quantidade de dados de treinamento. Portanto, transferir aprendizagem é fundamental para a implementação prática de modelos de aprendizado de máquina considerando o custo para treinamento sem conhecimento prévio. Usando o transferência de aprendizado, os operadores de rede podem resolver problemas novos, mas semelhantes a anteriores, de maneira mais econômica. No entanto, os efeitos negativos do conhecimento prévio sobre o desempenho do sistema necessitam maiores investigações. Ao se considerar as propostas em aprendizado profundo, identificam-se as tendências de adoção de técnicas mistas que combinam aprendizado profundo e por reforço. Além disso, são propostas redes de crença profunda (*deep belief networks*) que combinam diversas camadas de redes neurais, criando redes neurais ainda mais profundas, com mais camadas intermediárias profundas e diferentes ligações entre neurônios em cada camada.

Conclui-se que o processamento em fluxo de grandes massas de dados apresenta desafios significativos à aprendizagem profunda, incluindo grande escala, heterogeneidade, ruído na marcação dos rótulos e a distribuição não estacionária de atributos, entre outros. Assim, para realizar o potencial da analítica de grandes massas de dados em fluxo, há a necessidade de se enfrentar os desafios técnicos com propostas inovadoras e soluções transformadoras. Pesquisas em desafios impostos pela extração de conhecimento em grandes massas de dados em fluxo não são apenas oportunas, mas necessárias para diversos campos do conhecimento muito além de apenas a extração de conhecimento no gerenciamento de redes sem fio.

Referências

[Aaron et al., 2014] Aaron, B., Tamir, D. E., Rishe, N. D. e Kandel, A. (2014). Dynamic incremental k-means clustering. Em *2014 International Conference on Computational Science and Computational Intelligence*, volume 1, p. 308–313.

[Acer et al., 2015] Acer, U. G., Boran, A., Forlivesi, C., Liekens, W., Pérez-cruz, F. e Kawsar, F. (2015). Sensing wifi network for personal IoT analytics. Em *5th Internati-*

onal Conference on the Internet of Things, p. 104–111.

[Acer et al., 2016] Acer, U. G., Vanderhulst, G., Masshadi, A., Boran, A., Forlivesi, C., Scholl, P. M. e Kawsar, F. (2016). Capturing personal and crowd behavior with wi-fi analytics. Em *3rd International Workshop on Physical Analytics*, p. 43–48. ACM.

[Afanasyev et al., 2010] Afanasyev, M., Chen, T., Voelker, G. M. e Snoeren, A. C. (2010). Usage patterns in an urban wifi network. *IEEE/ACM Transactions on Networking (ToN)*, 18(5):1359–1372.

[Alessandrini et al., 2017] Alessandrini, A., Gioia, C., Sermi, F., Sofos, I., Tarchi, D. e Vespe, M. (2017). Wifi positioning and big data to monitor flows of people on a wide scale. Em *2017 European Navigation Conference (ENC)*, p. 322–328.

[Alsheikh et al., 2016] Alsheikh, M. A., Niyato, D., Lin, S., Tan, H.-P. e Han, Z. (2016). Mobile big data analytics using deep learning and apache spark. *IEEE Network*, 30(3):22–29.

[Andreoni Lopez et al., 2016] Andreoni Lopez, M., Lobato, A. G. P. e Duarte, O. C. M. B. (2016). A performance comparison of open-source stream processing platforms. Em *2016 IEEE Global Communications Conference (GLOBECOM)*, p. 1–6. IEEE.

[Andreoni Lopez et al., 2019] Andreoni Lopez, M., Mattos, D. M. F., Duarte, O. C. M. B. e Pujolle, G. (2019). A fast unsupervised preprocessing method for network monitoring. *Annals of Telecommunications*, 74.

[Andreoni Lopez et al., 2018] Andreoni Lopez, M., Sanz, I. J., Lobato, A. G. P., Mattos, D. M. F. e Duarte, O. C. M. B. (2018). Aprendizado de máquina em plataformas de processamento distribuído de fluxo: Análise e detecção de ameaças em tempo real. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, 2018:150–206.

[Armbrust et al., 2015] Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A. et al. (2015). Spark sql: Relational data processing in spark. Em *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, p. 1383–1394. ACM.

[Balbi et al., 2012] Balbi, H., Fernandes, N., Souza, F., Carrano, R., Albuquerque, C., Muchaluat-Saade, D. e Magalhaes, L. (2012). Centralized channel allocation algorithm for ieee 802.11 networks. Em *2012 Global Information Infrastructure and Networking Symposium (GIIS)*, p. 1–7.

[Blei e Smyth, 2017] Blei, D. M. e Smyth, P. (2017). Science and data science. *Proceedings of the National Academy of Sciences*, 114(33):8689–8692.

[Boutaba et al., 2018] Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F. e Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16.

[Bozkurt et al., 2015] Bozkurt, S., Elibol, G., Gunal, S. e Yayan, U. (2015). A comparative study on machine learning algorithms for indoor positioning. Em *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, p. 1–8. IEEE.

[Carbone et al., 2015a] Carbone, P., Fóra, G., Ewen, S., Haridi, S. e Tzoumas, K. (2015a). Lightweight asynchronous snapshots for distributed dataflows. *arXiv preprint arXiv:1506.08603*.

[Carbone et al., 2015b] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi†, S. e Tzoumas, K. (2015b). Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 4(34):28–38.

[Chen e Lin, 2014] Chen, X. e Lin, X. (2014). Big data deep learning: Challenges and perspectives. *IEEE Access*, 2:514–525.

[Chen et al., 2018] Chen, Y., Guizani, M., Zhang, Y., Wang, L., Crespi, N., Lee, G. M. e Wu, T. (2018). When traffic flow prediction and wireless big data analytics meet. *IEEE Network*.

[Chen e Qiu, 2011] Chen, Z. e Qiu, R. C. (2011). Cooperative spectrum sensing using q-learning with experimental validation. Em *2011 Proceedings of IEEE Southeastcon*, p. 405–408.

[Chilipirea et al., 2016] Chilipirea, C., Petre, A., Dobre, C. e van Steen, M. (2016). Presumably simple: Monitoring crowds using wifi. Em *Proceedings of the 17th IEEE International Conference on Mobile Data Management (MDM)*, volume 1, p. 220–225.

[Chintapalli et al., 2016] Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaugh, M., Liu, Z., Nusbaum, K., Patil, K., Peng, B. J. et al. (2016). Benchmarking streaming computation engines: Storm, flink and spark streaming. Em *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, p. 1789–1792. IEEE.

[Cici et al., 2015] Cici, B., Gjoka, M., Markopoulou, A. e Butts, C. T. (2015). On the decomposition of cell phone activity patterns and their connection with urban ecology. Em *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '15, p. 317–326.

[Cisco, 2017] Cisco, V. N. I. (2017). Global mobile data traffic forecast update, 2016–2021 white paper. *Document ID*, 1454457600805266.

[Costa et al., 2012] Costa, L. H. M. K., de Amorim, M. D., Campista, M. E. M., Rubinstein, M., Florissi, P. e Duarte, O. C. M. B. (2012). Grandes Massas de Dados na Nuvem: Desafios e Técnicas para Inovação. Em *SBRC 2012 - Minicursos*.

[Dean e Ghemawat, 2008] Dean, J. e Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[Deng, 2014] Deng, L. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3:e2.

[Deng e Yu, 2014] Deng, L. e Yu, D. (2014). Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4):197–387.

[Divgi e Chlebus, 2013] Divgi, G. e Chlebus, E. (2013). Characterization of user activity and traffic in a commercial nationwide wi-fi hotspot network: global and individual metrics. *Wireless Networks*, 19(7):1783–1805.

[Fan et al., 2016] Fan, Y., Chen, Y., Tung, K., Wu, K. e Chen, A. L. P. (2016). A framework for enabling user preference profiling through wi-fi logs. *IEEE Transactions on Knowledge and Data Engineering*, 28(3):592–603.

[Gaber, 2012] Gaber, M. M. (2012). Advances in data stream mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):79–85.

[Gama et al., 2014] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. e Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44.

[García et al., 2016] García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M. e Herrera, F. (2016). Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(1):9.

[Garofalakis et al., 2016] Garofalakis, M., Gehrke, J. e Rastogi, R. (2016). *Data Stream Management: Processing High-Speed Data Streams*. Springer.

[Godec et al., 2010] Godec, M., Leistner, C., Saffari, A. e Bischof, H. (2010). On-line random naive bayes for tracking. Em *2010 20th Int. Conference on Pattern Recognition*, p. 3545–3548.

[Goel et al., 2015] Goel, U., Wittie, M. P., Claffy, K. C. e Le, A. (2015). Survey of end-to-end mobile network measurement testbeds, tools, and services. *IEEE Communications Surveys & Tutorials*.

[Gonzalez et al., 2014] Gonzalez, J. E., Xin, R. S., Dave, A., Crankshaw, D., Franklin, M. J. e Stoica, I. (2014). GraphX: Graph processing in a distributed dataflow framework. Em *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, p. 599–613.

[Guo et al., 2018] Guo, J., Tan, Z.-H., Cho, S. H. e Zhang, G. (2018). Wireless personal communications: Machine learning for big data processing in mobile internet. *Wireless Personal Communications*, 102(3):2093–2098.

[Hadi et al., 2018] Hadi, M. S., Lawey, A. Q., El-Gorashi, T. E. e Elmirghani, J. M. (2018). Big data analytics for wireless and wired network design: A survey. *Computer Networks*, 132:180–199.

[Ham e Lee, 2014] Ham, Y. J. e Lee, H.-W. (2014). Big data preprocessing mechanism for analytics of mobile web log. *International Journal of Advances in Soft Computing & Its Applications*, 6(1).

[Han et al., 2011] Han, J., Pei, J. e Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

[Hofstede et al., 2014] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A. e Pras, A. (2014). Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*, 16(4):2037–2064.

[Hu et al., 2014] Hu, H., Wen, Y., Chua, T. e Li, X. (2014). Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access*, 2:652–687.

[Huang et al., 2016] Huang, H., Cai, Y. e Yu, H. (2016). Distributed-neuron-network based machine learning on smart-gateway network towards real-time indoor data analytics. Em *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, DATE '16, p. 720–725.

[Huang et al., 2014] Huang, W., Chen, Z., Dong, W., Li, H., Cao, B. e Cao, J. (2014). Mobile internet big data platform in china unicom. *Tsinghua Science and Technology*, 19(1):95–101.

[Iqbal e Soomro, 2015] Iqbal, M. H. e Soomro, T. R. (2015). Big data analysis: Apache storm perspective. *Int. journal of computer trends and technology*, 19(1):9–14.

[Isard et al., 2007] Isard, M., Budiu, M., Yu, Y., Birrell, A. e Fetterly, D. (2007). Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS operating systems review*, 41(3):59–72.

[Jang et al., 2017] Jang, R., Cho, D., Noh, Y. e Nyang, D. (2017). Rflow+: An sdn-based wlan monitoring and management framework. Em *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, p. 1–9.

[Jiang et al., 2017] Jiang, C., Zhang, H., Ren, Y., Han, Z., Chen, K.-C. e Hanzo, L. (2017). Machine learning paradigms for next-generation wireless networks. *IEEE Wireless Communications*, 24(2):98–105.

[Jordaney et al., 2017] Jordaney, R., Sharad, K., Dash, S. K., Wang, Z., Papini, D., Nouretdinov, I. e Cavallaro, L. (2017). Transcend: Detecting concept drift in malware classification models. Em *PROCEEDINGS OF THE 26TH USENIX SECURITY SYMPOSIUM*, p. 625–642. USENIX Association.

[Kim et al., 2010] Kim, S. K., McMahon, P. L. e Olukotun, K. (2010). A large-scale architecture for restricted boltzmann machines. Em *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, p. 201–208.

[Kwon et al., 2017] Kwon, D., Kim, H., Kim, J., Suh, S. C., Kim, I. e Kim, K. J. (2017). A survey of deep learning-based network anomaly detection. *Cluster Computing*.

[Lee et al., 2005] Lee, G. M., Liu, H., Yoon, Y. e Zhang, Y. (2005). Improving sketch reconstruction accuracy using linear least squares method. Em *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, p. 24–24, Berkeley, CA, USA. USENIX Association.

[Leung e Kim, 2003] Leung, K. K. e Kim, B. J. (2003). Frequency assignment for IEEE 802.11 wireless networks. Em *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, volume 3, p. 1422–1426 Vol.3.

[Li et al., 2013] Li, B., Springer, J., Bebis, G. e Hadi Gunes, M. (2013). Review: A survey of network flow applications. *Journal of Network and Computer Applications*, 36(2):567–581.

[Li, 2018] Li, Y. (2018). Deep Reinforcement Learning. *arXiv e-prints*, p. arXiv:1810.06339.

[Liebchen et al., 2007] Liebchen, G., Twala, B., Shepperd, M., Cartwright, M. e Stephens, M. (2007). Filtering, robust filtering, polishing: Techniques for addressing quality in software data. Em *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, p. 99–106. IEEE.

[Lin et al., 2017] Lin, F. Y. S., Wen, Y. F., Fang, L. W. e Hsiao, C. H. (2017). Resource allocation and multisession routing algorithms in coordinated multipoint wireless communication networks. *IEEE Systems Journal*, PP(99):1–12.

[Liu e Yoo, 2017] Liu, Y. e Yoo, S. (2017). Dynamic resource allocation using reinforcement learning for lte-u and wifi in the unlicensed spectrum. Em *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, p. 471–475.

[Lobato et al., 2018] Lobato, A. G. P., Andreoni Lopez, M., Sanz, I. J., Cardenas, A. A., Duarte, O. C. M. B. e Pujolle, G. (2018). An adaptive real-time architecture for zero-day threat detection. Em *2018 IEEE International Conference on Communications (ICC)*, p. 1–6.

[Low et al., 2012] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A. e Hellerstein, J. M. (2012). Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727.

[Low et al., 2014] Low, Y., Gonzalez, J. E., Kyrola, A., Bickson, D., Guestrin, C. E. e Hellerstein, J. (2014). Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041*.

[Luiz et al., 2015] Luiz, T. A., Freitas, A. R. e Guimarães, F. G. (2015). A new perspective on channel allocation in wlan: Considering the total marginal utility of the connections for the users. Em *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, p. 879–886, New York, NY, USA. ACM.

[Magalhães e Mattos, 2018] Magalhães, L. C. S. e Mattos, D. M. F. (2018). Caracterização do uso de uma rede sem fio de grande porte distribuída por uma ampla Área. *XVII*

Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance - CSBC 2018), 17(1/2018).

[Malewicz et al., 2010] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N. e Czajkowski, G. (2010). Pregel: a system for large-scale graph processing. Em *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, p. 135–146. ACM.

[Mattos et al., 2018] Mattos, D. M. F., Velloso, P. B. e Duarte, O. C. M. B. (2018). Uma infraestrutura Ágil e efetiva de virtualização de funções de rede para a internet das coisas. Em *XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2018*.

[Maturi et al., 2017] Maturi, F., Gringoli, F. e Cigno, R. L. (2017). A dynamic and autonomous channel selection strategy for interference avoidance in 802.11. Em *2017 13th Annual Conference on Wireless On-demand Network Systems and Services(WONS)*, p. 1–8.

[Meng et al., 2016] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S. et al. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241.

[Monteiro et al., 2016] Monteiro, A., Souto, E., Pazzi, R. e Kiljander, J. (2016). Atribuição dinâmica de canais em redes sem fio não coordenadas ieee 802.11, baseada em fatores de sobreposição e intensidade de sinal. Em *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2016*.

[Moura et al., 2019] Moura, H. D., Macedo, D. F. e Vieira, M. A. M. (2019). Automatic quality of experience management for wlan networks using multi-armed bandit. Em *IFIP / IEEE International Symposium on Integrated Network Management*, p. 1–10. IEEE/IFIP.

[Qiu et al., 2016] Qiu, J., Wu, Q., Ding, G., Xu, Y. e Feng, S. (2016). A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016(1):67.

[Shin et al., 2004] Shin, M., Mishra, A. e Arbaugh, W. A. (2004). Improving the latency of 802.11 hand-offs using neighbor graphs. Em *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services*, MobiSys '04, p. 70–83, New York, NY, USA. ACM.

[Song et al., 2010] Song, C., Qu, Z., Blumm, N. e Barabási, A.-L. (2010). Limits of predictability in human mobility. *Science*, 327(5968):1018–1021.

[Sun et al., 2018] Sun, Y., Peng, M., Zhou, Y., Huang, Y. e Mao, S. (2018). Application of machine learning in wireless networks: Key techniques and open issues. Relatório técnico, arXiv preprint arXiv:1809.08707. Online. `https://arxiv.org/abs/1809.08707`. Acessado em 22/03/2019.

[Sze et al., 2017] Sze, V., Chen, Y., Yang, T. e Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329.

[Tabrizi et al., 2011] Tabrizi, H., Farhadi, G. e Cioffi, J. (2011). A learning-based network selection method in heterogeneous wireless systems. Em *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, p. 1–5.

[Tannahill e Jamshidi, 2014] Tannahill, B. K. e Jamshidi, M. (2014). System of systems and big data analytics–bridging the gap. *Computers & Electrical Engineering*, 40(1):2–15.

[Tesauro, 2007] Tesauro, G. (2007). Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1):22–30.

[Toch et al., 2018] Toch, E., Lerner, B., Ben-Zion, E. e Ben-Gal, I. (2018). Analyzing large-scale human mobility data: a survey of machine learning methods and applications. *Knowledge and Information Systems*, p. 1–23.

[Tsai et al., 2015] Tsai, C.-W., Lai, C.-F., Chao, H.-C. e Vasilakos, A. V. (2015). Big data analytics: a survey. *Journal of Big Data*, 2(1):21.

[Turgut et al., 2019] Turgut, Z., Üstebay, S., Zeynep Gürkaş Aydın, G. e Sertbaş, A. (2019). Deep learning in indoor localization using WiFi. Em Boyaci, A., Ekti, A. R., Aydin, M. A. e Yarkan, S., editors, *International Telecommunications Conference*, p. 101–110, Singapore. Springer.

[Wang et al., 2018] Wang, F., Gong, W., Liu, J. e Wu, K. (2018). Channel selective activity recognition with WiFi: A deep learning approach exploring wideband information. *IEEE Transactions on Network Science and Engineering*, p. 1–1.

[Wang et al., 2013] Wang, S., Minku, L. L., Ghezzi, D., Caltabiano, D., Tino, P. e Yao, X. (2013). Concept drift detection for online class imbalance learning. Em *The 2013 Int. Joint Conference on Neural Networks (IJCNN)*, p. 1–10.

[Wang et al., 2018] Wang, T., Yang, Q., Tan, K., Zhang, J., Liew, S. C. e Zhang, S. (2018). Dcap: Improving the capacity of wifi networks with distributed cooperative access points. *IEEE Transactions on Mobile Computing*, 17(2):320–333.

[Wang et al., 2017] Wang, X., Gao, L. e Mao, S. (2017). BiLoc: Bi-Modal Deep Learning for Indoor Localization With Commodity 5GHz WiFi. *IEEE Access*, 5:4209–4220.

[Wang et al., 2018] Wang, X., Wang, X. e Mao, S. (2018). RF sensing in the internet of things: A general deep learning framework. *IEEE Communications Magazine*, 56(9):62–67.

[Warneke e Kao, 2009] Warneke, D. e Kao, O. (2009). Nephele: efficient parallel data processing in the cloud. Em *Proceedings of the 2nd workshop on many-task computing on grids and supercomputers*, p. 8. ACM.

[Xin et al., 2013] Xin, R. S., Rosen, J., Zaharia, M., Franklin, M. J., Shenker, S. e Stoica, I. (2013). Shark: SQL and rich analytics at scale. Em *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*, p. 13–24. ACM.

[Xu et al., 2017] Xu, G., Gao, S., Daneshmand, M., Wang, C. e Liu, Y. (2017). A survey for mobility big data analytics for geolocation prediction. *IEEE Wireless Communications*, 24(1):111–119.

[Zaharia et al., 2012a] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S. e Stoica, I. (2012a). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Em *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, p. 2–2. USENIX Association.

[Zaharia et al., 2013] Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S. e Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. Em *XXIV ACM Symposium on Operating Systems Principles*, p. 423–438. ACM.

[Zaharia et al., 2012b] Zaharia, M., Das, T., Li, H., Shenker, S. e Stoica, I. (2012b). Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, p. 10–10.

[Zeng et al., 2015] Zeng, Y., Pathak, P. H. e Mohapatra, P. (2015). Analyzing shopper's behavior through wifi signals. Em *Proceedings of the 2Nd Workshop on Physical Analytics*, WPA '15, p. 13–18.

[Zeng et al., 2016] Zeng, Y., Pathak, P. H. e Mohapatra, P. (2016). Wiwho: Wifi-based person identification in smart spaces. Em *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, IPSN '16, p. 4:1–4:12.

[Zhang et al., 2016] Zhang, X., Wang, C., Li, Z., Zhu, J., Shi, W. e Wang, Q. (2016). Exploring the sequential usage patterns of mobile internet services based on markov models. *Electronic Commerce Research and Applications*, 17:1 – 11.

Capítulo 5

Introdução à criptografia para administradores de sistemas com TLS, `OpenSSL` e Apache `mod_ssl`

Alexandre Braga (UNICAMP) e Ricardo Dahab (UNICAMP)

Abstract

The incorrect use of cryptographic infrastructures is one of the most common sources of cryptography-related security issues: avoidable errors related to configuration of algorithms and protocols become recurrent, often resulting in exploitable vulnerabilities and actual attacks on computing systems. Hence, there is a growing demand for practical, real-world guidance on how to avoid incorrect cryptographic configurations that lead to exploitable vulnerabilities in computer networks and distributed systems. This chapter addresses the use of cryptography by students and IT professionals with little experience in information security and cryptography. The text covers `OpenSSL`, the TLS protocol (including the new version 1.3 launched in August 2018), and security settings for the `mod_ssl` module integrated to the Apache web server.

Resumo

O uso incorreto de infraestruturas criptográficas é uma das fontes mais comuns de problemas de segurança relacionados à criptografia, quando diversos erros evitáveis na configuração de algoritmos e de protocolos se tornam recorrentes, resultando frequentemente em vulnerabilidades exploráveis em ataques reais aos sistemas de computação. Por isso, há uma demanda crescente por orientações práticas, do mundo real, sobre como evitar configurações incorretas da criptografia que levam a vulnerabilidades exploráveis em redes de computadores e sistemas distribuídos. Este capítulo aborda a utilização de criptografia por estudantes e profissionais de TI com pouca experiência em segurança da informação e criptografia. O minicurso cobre o `OpenSSL`, o protocolo TLS (incluindo a nova versão 1.3 de agosto de 2018) e as configurações seguras do módulo `mod_ssl` do servidor web Apache.

5.1. Introdução

O uso incorreto de infraestruturas criptográficas é uma das fontes mais comuns de problemas de segurança relacionados à criptografia, em que diversos erros evitáveis na configuração de algoritmos e de protocolos se tornaram recorrentes, resultando frequentemente em vulnerabilidades exploráveis em ataques reais aos sistemas de computação. No contexto da indústria de Tecnologia da Informação (TI), há uma demanda crescente pelos aspectos práticos, do mundo real, relacionados às configurações corretas da criptografia que evitem vulnerabilidades exploráveis em redes de computadores e sistemas distribuídos.

Este capítulo cobre, em nível introdutório, o `OpenSSL`, o protocolo TLS, incluindo a nova versão (v1.3) de agosto de 2018, e as configurações do módulo `mod_ssl` do servidor web Apache. O software `OpenSSL` é a principal ferramenta de criptografia escolhida para a realização de exercícios e demonstrações por que possui uma *Command Line Interface* (CLI) estável e que tem sido usada por profissionais de TI há muito tempo. Além disso, o protocolo TLS e o `OpenSSL` compõem a infraestrutura criptográfica utilizada por diversos softwares na base dos sistemas da Internet, tais como servidores web (incluindo o `mod_ssl` do Apache), servidores de aplicação, contêineres de software, plataformas móveis e até dispositivos da Internet das Coisas, entre outros.

O restante do texto está organizado da seguinte forma. A Seção 5.2 conta uma breve história da evolução do SSL, oferecendo uma visão geral do funcionamento deste protocolo. A Seção 5.3 recapitula os conceitos fundamentais da criptografia. A Seção 5.4 está organizada como um tutorial de comandos do `OpenSSL`. Já a Seção 5.5 mostra como testar a segurança criptográfica do servidor Apache, enquanto a Seção 5.6 descreve as configurações seguras e inseguras de SSL. Finalmente, a Seção 5.7 faz as considerações finais.

5.2. Breve história do surgimento e evolução do SSL

Security Sockets Layer, ou simplesmente SSL (camada de segurança de *sockets*, em tradução livre) é o protocolo para transporte de dados protegidos com criptografia sobre os *sockets* do protocolo TCP. Em relação a pilha de protocolos TCP/IP, a camada de *socket* seguro fica logo acima da camada de transporte. De fato, as aplicações (na camada de aplicação) podem usar os *sockets* seguros como se fossem um serviço oferecido pela camada de transporte.

O SSL permite que a comunicação pela Internet entre aplicações cliente/servidor possa ser protegida contra monitoramento, adulteração de conteúdo e falsificação de mensagens. O objetivo principal do SSL sempre foi proporcionar um canal de comunicação seguro entre partes comunicantes. Tradicionalmente, livros de segurança em redes de computadores [Stallings 2003] abordam o SSL, havendo também livros específicos [Chandra et al. 2002, Ristic 2015] sobre a implementação de código aberto mais conhecida deste protocolo, o `OpenSSL` [OpenSSL.org].

A história do protocolo SSL se confunde com a história do comércio eletrônico na Internet. Na última década do século passado, a empresa Netscape [Wikipedia 2018] dominava tanto o mercado de software de navegadores web (*browsers*) quanto o de servidores Web. Em um modelo de negócios inovador para a época, a Netscape distribuía gratuitamente o *browser*, mas vendia o seu servidor web, o único na época que possuia proteções de sigilo contra monitoramento e interceptação da comunicação. A segurança da comunicação era garantida por

um protocolo criptográfico na camada de transporte do TCP/IP que ficou conhecido como SSL.

O SSL tornou possíveis as transações sigilosas entre o*browser* web na máquina do usuário e o servidor web. O uso do SSL sempre foi praticamente transparente para o usuário final, o que facilitou muito a ampla adoção deste protocolo como padrão de fato para segurança na Internet. Por exemplo, com o SSL, os números de cartões de crédito puderam transitar pelas redes abertas com sigilo, viabilizando o comércio eletrônico como acontece hoje em dia. A última versão do SSL original foi a terceira (SSLv3), que trouxe as seguintes características de segurança disponíveis até hoje:

- Autenticação do servidor com assinaturas digitais e certificados X.509, com os algoritmos RSA, DSA e ECDSA, ou uma chave simétrica pré-configurada;
- Autenticação (opcional) do cliente com assinaturas digitais e certificados X.509;
- Sigilo com encriptação da informação trocada entre cliente e servidor;
- Integridade (com uso de MACs) da informação trocada entre cliente e servidor.

Uma vez que o SSL se tornou um padrão de fato, com o seu uso bastante difundido, surgiu a necessidade de padronização da tecnologia. Foi a terceira versão do SSL (SSLv3) que serviu de base para o *Transport Layer Security* (TLS), um protocolo padronizado pelo *Internet Engineering Task Force* (IETF) com o objetivo de substituir o formato proprietário do SSL original. Por razões históricas, o protocolo TLS também é conhecido como SSL/TLS.

O SSL/TLS possui dois sub-protocolos: a saudação (*handshake*), que autentica as partes e negocia os parâmetros de segurança criptográfica da comunicação, e o protocolo de registro, que usa os parâmetros escolhidos na saudação para proteger o tráfego entre as partes comunicantes. A Figura 5.1 ilustra o funcionamento do *handshake* do SSL/TLS v1.2. As etapas do diálogo de saudação entre cliente e servidor são detalhadas e enumeradas na Figura 5.1. O objetivo principal deste diálogo é o estabelecimento de uma chave de sessão (uma chave criptográfica secreta, simétrica e temporária). O diálogo de saudação pode ser dividido em quatro partes:

1. Negociação de parâmetros (versão, ID de sessão, algoritmos criptográficos e compressão);
2. Envio do certificado do servidor e solicitação opcional do certificado do cliente;
3. Envio do certificado do cliente, se solicitado;
4. Escolha de algoritmos criptográficos e finalização;

A implementação do *handshake* em duas rodadas é considerada [Sullivan 2018] uma causa de perda de desempenho na execução do protocolo TLSv1.2 e foi otimizada na versão TLSv1.3. Após a realização do protocolo de *handshake*, entra em ação o protocolo de registro que realiza a troca, entre cliente e servidor, de informação encriptada e segmentada em blocos chamados de registros.

5.2.1. Versões do SSL/TLS e suas vulnerabilidades

Além do SSLv3 que deu origem ao TLSv1, vale ainda mencionar outras versões do SSL/TLS que se destacam por possuírem características específicas e vulnerabilidades conhecidas. Em linhas gerais, existem seis versões em uso do SSL/TLS que ainda podem ser encontradas em

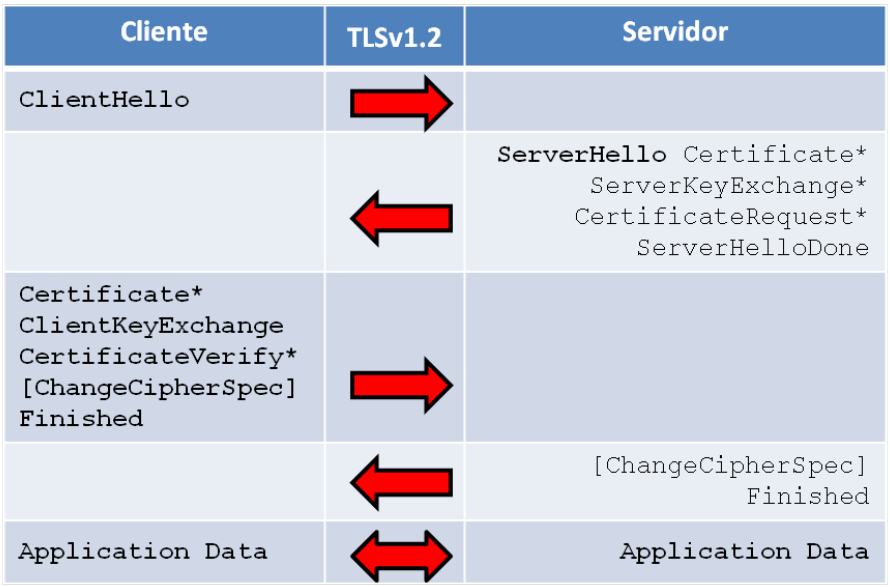


Figura 5.1. Estabelecimento de sessão e envio de mensagens SSL/TLS v1.2.

implantações reais: SSLv2, SSLv3, TLSv1.0, TLSv1.1, TLSv1.2 e TLSv1.3. As características gerais de cada uma delas são descritas a seguir:

- SSLv2 é considerado inseguro e não deve mais ser usado. Esta versão é vulnerável a ataques conhecidos, tais como o BEAST (Browser Exploit against SSL/TLS) e o DROWN (Decrypting RSA with Obsolete and Weakened eNcryption) [Aviram et al. 2016], e outras vulnerabilidades como *Cipher Suite Rollback* e *ChangeCipherSpec Message Drop*.
- SSLv3 é obsoleto e não deve mais ser utilizado. O HTTPS sobre SSLv3 é vulnerável aos ataques POODLE (Padding Oracle on Downgraded Legacy Encryption) [Möller et al. 2014] e *Lucky 13* [Fardan and Paterson 2013] e sofre de vulnerabilidades como o *Version Rollback* e o *Key Exchange Algorithm Confusion*.
- TLSv1.0 (RFC 2246) ainda é usada em sistemas legados. TLSv1.0 é vulnerável aos ataques BEAST e *Lucky 13* [Fardan and Paterson 2013]. Esta versão inicial do TLS também é vulnerável aos ataques de negação de serviço e que exploram falhas na renegociação. Padrões de segurança como o PCI-DSS recomendam que esta versão seja abandonada.
- TLSv1.1 (RFC 4346) é uma versão relativamente recente e que não apresenta vulnerabilidades conhecidas sem mitigação, mas ainda oferece algoritmos criptográficos antigos.
- TLSv1.2 (RFC 5246) era a versão atual até Agosto de 2018, quando foi lançada a versão nova, e já oferece esquemas criptográficos novos com encriptação autenticada.
- TLSv1.3 (RFC 8446) foi lançado na forma final em Agosto de 2018 [Rescorla 2018] e representa o rompimento definitivo com o passado pela eliminação de diversas características inseguras ou obsoletas mantidas até a versão anterior por compatibilidade com legados.

Há testes específicos para implantações do SSL/TLS [Ristic 2015, Eldewahi et al. 2015, OWASP 2015]. Outros ataques contra SSL/TLS bastante conhecidos são o CRIME (*Compression Ratio info-leak Made Easy*)[CRI 2012], o BREACH (*Browser Reconnaissance and*

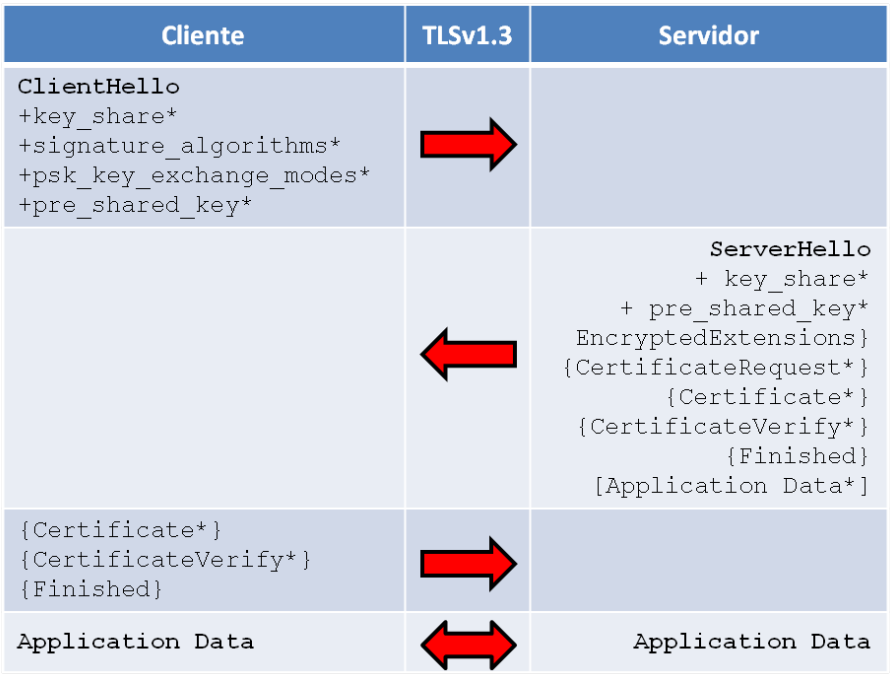


Figura 5.2. Estabelecimento de sessão e envio de mensagens SSL/TLS v1.3.

Exfiltration via Adaptive Compression of Hypertext) [BRE 2013, Gluck et al. 2013], o *Bleichenbacher Attack on PKCS#1* [Bleichenbacher 1998], os ataques ao RC4 [AlFardan et al. 2013], o Heartbleed [Synopsys 2014], e o LogJam [Adrian et al. 2015, Log 2016], que afeta o acordo de chaves.

5.2.2. A versão 1.3 do SSL/TLS

A versão 1.3 do TLS [Rescorla 2018] foi lançada na forma final em Agosto de 2018. Este texto inclui um breve relato dos avanços apresentados por esta nova versão em relação a sua antecessora. A nova versão 1.3 atende a quatro objetivos de projeto [Sullivan 2018]: redução da latência do *handshake*, encriptação de partes do *handshake*, aumento da robustez contra ataques e remoção de funções legadas. A Figura 5.2 ilustra o *handshake* do SSL/TLS na versão 1.3, onde se observa que o protocolo é mais curto que o da versão anterior. As principais diferenças entre o TLSv1.2 e o TLSv1.3 são as seguintes [Rescorla 2018]:

- A lista de opções de algoritmos simétricos foi bastante reduzida e só contém encriptação autenticada (*Authenticated Encryption with Additional Data* - AEAD), já a encriptação de blocos em modo CBC e o RC4 foram removidos da versão;
- O conceito de *suite* criptográfica foi simplificado e agora separa os mecanismos de autenticação e troca de chaves dos mecanismos de encriptação, *hash* usado na derivação de chaves e MACs;
- O *handshake* simplificado, em uma única rodada, acelera a saudação; além disso, o *handshake* sem rodadas (sem negociações, só comunicação de escolhas pré-definidas) foi incluído para atender às aplicações com restrições de tempo ainda maiores;

- RSA e DH com parâmetros estáticos foram removidos, de modo que todas as *suites* criptográficas desta versão possuem *forward secrecy*; porém, só o DHE/ECDHE é usado para troca de chaves;
- Todas as mensagens do *handshake* depois do "ServerHello"agora são encriptadas e assinadas pelo servidor, para proteção da comunicação contra ataques práticos, como o FREAK e o LogJam, e outros considerados teóricos, como a troca maliciosa de curvas elípticas (*curveswap*);
- A criptografia de curvas elípticas passou a fazer parte da especificação principal protocolo, mas a negociação de formatos de representação de pontos da curva foi removida, para incluir apenas um formato não negociável;
- Em termos de criptografia de chaves públicas, o RSA-PSS é o único *padding* do RSA disponível, já o *padding* inseguro do padrão PKCS#1 v1.5 foi removido, assim como também foram removidos o DSA e a compressão;
- DH com parâmetros efêmeros customizados não está mais disponível nesta versão, quer possui apenas um número fixo de opções seguras, evitando a parametrização fraca do DH que leva, por exemplo, ao ataque LogJam [Log 2016];
- A Mensagem "ChangeCipherSpec"foi removida do *handshake*, simplificando o protocolo e aumentando a robustez contra ataques, enquanto o *handshake* com chave pré-compartilhada (*pre-shared key* - PSK) foi simplificado e aumenta a robustez contra ataques.

5.2.3. Limitações do SSL/TLS

O SSL/TLS, como todo protocolo de segurança, tem limitações e não resolve todos os problemas de segurança de transações eletrônicas na Internet. Em particular, o SSL/TLS só protege a comunicação entre o software navegador web e o servidor HTTPS. Por exemplo, uma vez que a informação encriptada de um cartão de crédito chega ao servidor HTTPS do lojista, ela precisa ser decriptada para que o pagamento seja efetivado junto à instituição financeira (banco ou operadora de cartão de crédito). Neste momento, as informações do cartão ficam expostas ao lojista e aos seus funcionários. O SSL/TLS não resolve este problema, pois ele trata apenas da comunicação entre *browser* e servidor web. Outros protocolos de segurança devem ser usados para proteger as informações de pagamento.

Além disso, o SSL/TLS é independente de aplicação e, por isto, sua documentação não especifica como ele deve ser adicionado a um protocolo de aplicação. Assim sendo, decisões de projeto importantes são deixadas a cargo dos implementadores de protocolos de aplicação, tais como em que momento o *handshake* deve ser iniciado ou como interpretar a autenticação dos certificados digitais trocados entre as partes. Esta situação abre espaço para a inclusão de defeitos de implementação que levam a vulnerabilidades exploráveis.

Finalmente, surgiram recentemente ataques que exploram a confiança já estabelecida em *sites* web que possuem conexões SSL/TLS legítimas. O abuso da confiança depositada por usuários na conexão SSL/TLS entre *browser* e servidor web ocorre quando uma conexão SSL/TLS legítima é usada em ataques. Atualmente, mesmo sites web maliciosos, utilizados, por exemplo, em ataques de *phishing*, podem ter certificados SSL legítimos em seus servidores. Isto se deve ao fato de autoridades certificadores emitirem, de forma simplificada e na Internet, certificados

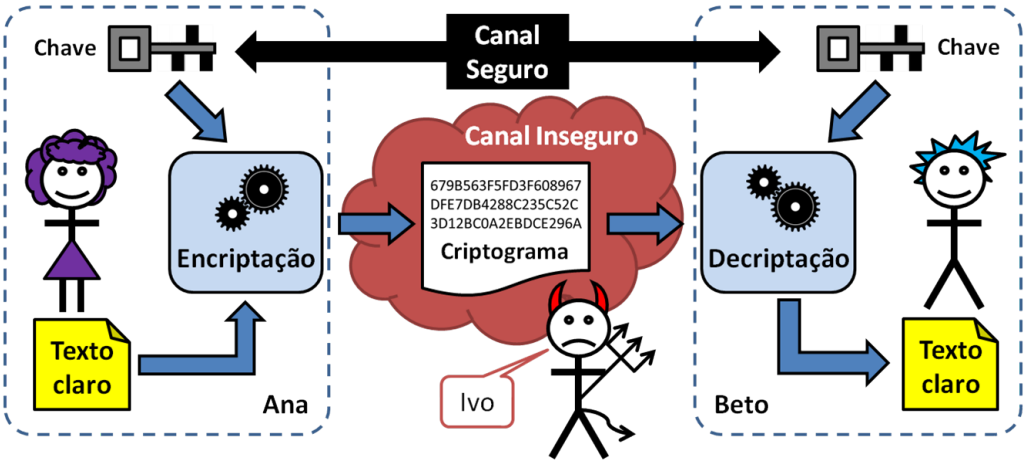


Figura 5.3. Sistema criptográfico (de [Braga and Dahab 2015]).

gratuitos e de validade curta. Estes certificados, mesmo com validade reduzida, ainda podem ser usados por tempo suficiente para viabilizar ataques de engenharia social [Calvo 2018].

5.3. Conceitos fundamentais de criptografia

Historicamente associada ao sigilo, a criptografia moderna também oferece serviços para autenticidade, integridade e irrefutabilidade. Esta seção recapitula os conceitos de serviços criptográficos comumente encontrados em redes de computadores e sistemas de TI. Em linhas gerais, os objetivos e funções da criptografia são os seguintes:

1. Confidencialidade (ou sigilo) é obtida com o uso da criptografia para manter a informação secreta, confidencial. Protocolos de comunicação segura com encriptação de dados em trânsito são exemplos de confidencialidade.
2. Autenticidade é obtida com o uso da criptografia para validar a autoria de uma informação. Um exemplo de autenticação é o uso de assinaturas digitais para verificar a autoria de uma mensagem de texto ou de um documento eletrônico.
3. Integridade é obtida com o uso da criptografia para garantir que o dado não foi modificado desde a sua criação. Valores *hash* usados na validação de arquivos binários são exemplos de mecanismos para verificação de integridade de dados.
4. Irrefutabilidade ou não-repudiação é obtida pelo uso da criptografia para garantir que o autor da informação autêntica não possa negar para um terceiro a autoria desta informação. Assinaturas digitais podem ser usadas na garantia de irrefutabilidade.

Na prática, os objetivos são combinados e as funções são usadas juntas. Por exemplo, em um aplicativo de mensageiria instantânea, a troca de mensagens entre as partes comunicantes pode ser encriptada e assinada digitalmente, garantindo tanto a confidencialidade quanto a autenticidade e a integridade do diálogo.

A Figura 5.3 (adaptada de [Braga and Dahab 2015]) mostra um sistema criptográfico e seus elementos estruturais com três personagens: Ana, a remetente das mensagens; Beto, o destinatário das mensagens; e Ivo, o adversário. As mensagens passam por um canal de

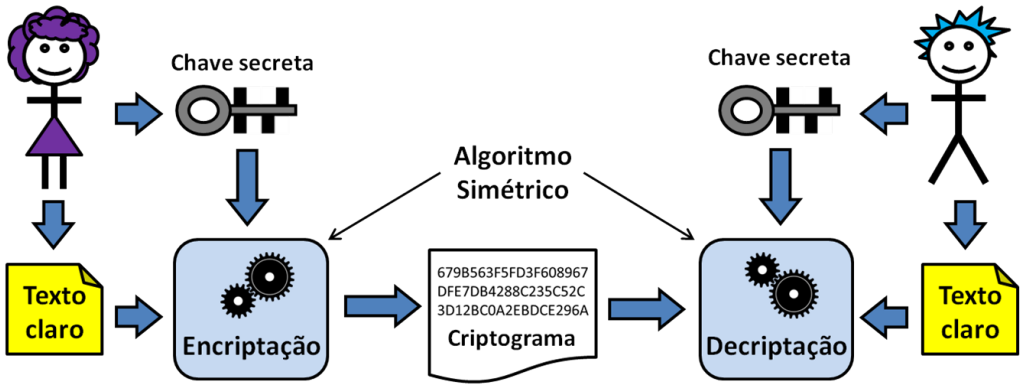


Figura 5.4. Sistema criptográfico simétrico (de [Braga and Dahab 2015]).

comunicação inseguro e controlado por Ivo. O algoritmo criptográfico é usado para transformar o texto claro (legível por qualquer um) em texto encriptado (o criptograma legível apenas por Ana e Beto) e vice-versa.

A chave criptográfica é o parâmetro de configuração do algoritmo que viabiliza a recuperação de um texto claro a partir do texto encriptado. Ana e Beto usam uma chave criptográfica conhecida apenas por eles e compartilhada (ou combinada) por um canal seguro diferenciado. A segurança do sistema criptográfico reside no segredo da chave e não no segredo do algoritmo criptográfico. Assim, se um algoritmo de boa reputação é usado, a qualidade da implementação deste algoritmo e o tamanho da chave criptográfica determinam a dificuldade em violar a encriptação da mensagem.

Existem dois tipos de sistemas criptográficos, comumente conhecidos como criptografia de chave secreta (ou simétrica) e criptografia de chave pública (ou assimétrica). Na criptografia de chave secreta, uma única chave é usada para encriptar e decriptar a informação. Na criptografia de chave pública, duas chaves são necessárias. Uma chave é usada para encriptar; a outra chave, diferente, é usada para decriptar a informação. O desempenho (a velocidade) da criptografia simétrica é geralmente superior a da criptografia assimétrica, no mesmo nível de segurança.

Por causa das questões administrativas de distribuição de chaves, a criptografia de chave pública é recomendada para grupos grandes e ambientes dinâmicos e abertos. Por outro lado, a criptografia de chave secreta é recomendada para grupos pequenos onde as partes comunicantes já estão bem definidas. Conforme descrito adiante no texto, uma solução amplamente utilizada pelos protocolos de comunicação segura na Internet usa uma combinação das tecnologias simétrica e assimétrica, chamada de sistemas criptográficos híbridos.

5.3.1. Criptografia simétrica (de chave secreta)

A Figura 5.4 ilustra os passos de encriptação e decriptação com algoritmos simétricos de chave secreta. Apenas a chave usada na encriptação pode decriptar corretamente a informação. Por isso, esta chave deve ser protegida e guardada em segredo; daí o nome de chave secreta. Este sistema criptográfico poderia ser diretamente utilizado para encriptação bidirecional com a mesma chave. Porém, na prática, diversos detalhes de implementação dificultam a utilização segura da mesma chave para encriptação nas duas direções do canal de comunicação.

Na criptografia simétrica, a chave secreta deve ser conhecida por todos aqueles que

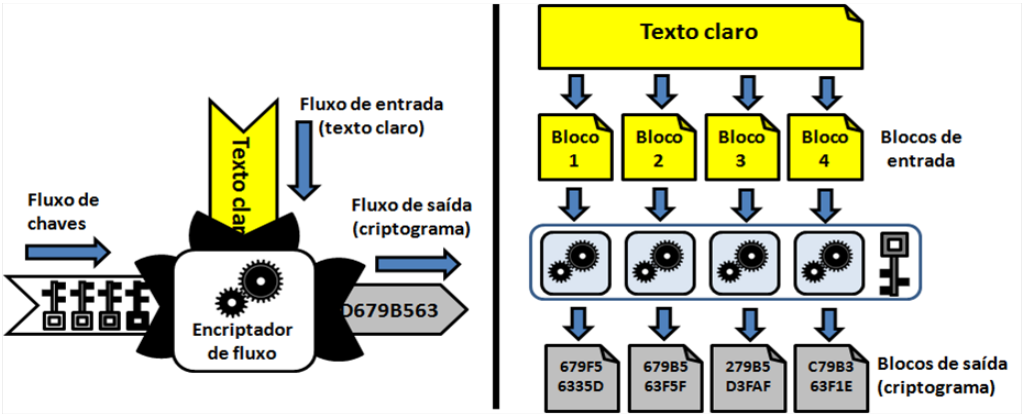


Figura 5.5. Encriptação simétrica de fluxo e de blocos (de [Braga and Dahab 2015]).

precisam decriptar a informação encriptada com ela. Um dos problemas práticos é o compartilhamento ou distribuição segura da chave secreta. A distribuição de chaves é um aspecto importante da criptografia de chave secreta. Apesar das dificuldades de distribuição de chaves, a criptografia de chave secreta é muito usada. Suas dificuldades aparentes podem ser contornadas pela combinação desta tecnologia com a criptografia de chave pública, originando sistemas criptográficos híbridos.

Existem duas categorias de algoritmos simétricos de encriptação (Figura 5.5): de fluxo e de bloco. Na encriptação de blocos, o texto claro é quebrado em blocos (de bits) de tamanho fixo. A encriptação ocorre sobre blocos individualmente e produz criptogramas em blocos também. O tamanho da chave criptográfica é geralmente um múltiplo do tamanho do bloco. A encriptação de fluxo trabalha sobre sequências (de bits). A sequência ou fluxo de entrada é transformado continuamente na sequência ou fluxo de saída, bit a bit. Neste caso, é importante que a chave criptográfica seja uma sequência de bits pelo menos do mesmo tamanho do fluxo de entrada. Na prática, os bits da chave podem ser produzidos por um gerador de sequências de bits pseudoaleatórias, a partir de uma chave mestra de tamanho fixo. Exemplos de algoritmos criptográficos simétricos comumente usados atualmente são o AES, o SERPENT, e o 3DES.

Os algoritmos simétricos de bloco trabalham sobre dados com tamanho igual a um múltiplo inteiro do tamanho do bloco. Já o texto claro tem tamanho livre. Para obter flexibilidade no texto claro, é necessário um mecanismo de preenchimento de blocos incompletos a fim de que o tamanho do texto claro seja múltiplo do tamanho do bloco. Chama-se *padding* (preenchimento) o completamento do texto claro para que ele seja múltiplo do tamanho do bloco do algoritmo de encriptação.

Os modos de operação da encriptação de blocos combinam de maneiras diferentes os blocos do texto claro (ou do criptograma) com a chave criptográfica, para produzir encadeamentos de blocos encriptados característicos de cada modo de operação. Assim, o mesmo texto claro, encriptado com a mesma chave, mas usando modos de operação diferentes, resulta em criptogramas diferentes. Os modos de operação mais comuns são ECB, CBC, CFB, OFB e CTR [NIST 2001]. Os modos ECB e CBC são típicos da encriptação de blocos. Os modos CFB, OFB e CTR se comportam como na encriptação de fluxo. Os modos de operação, exceto o ECB, evitam a ocorrência de padrões identificáveis nos bits do criptograma. Os modos de

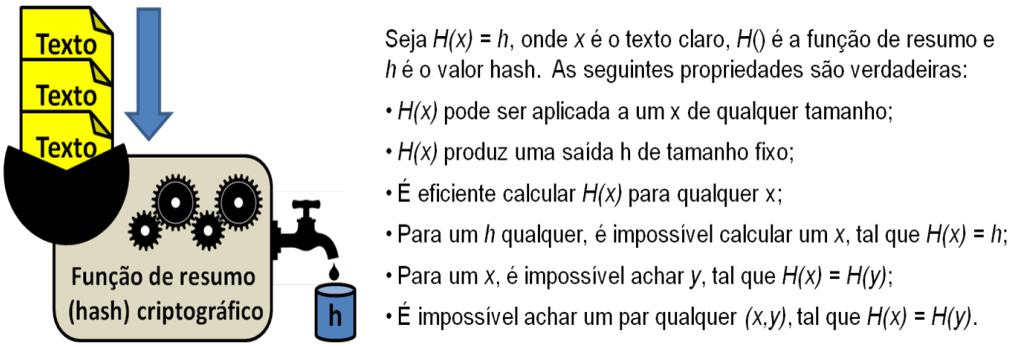


Figura 5.6. Funções de resumo (hash) criptográfico (de [Braga and Dahab 2015]).

operação são diferentes em relação à recuperação de erros e à perda de sincronismo.

5.3.2. Resumo criptográfico (*hash*) e Código de Autenticação de Mensagem (MAC)

O uso de encriptação somente não é capaz de garantir que o criptograma não foi corrompido, maliciosamente modificado, ou até mesmo completamente substituído por Ivo, quando em trânsito de Ana até Beto. Em uma solução incial simples e incompleta desta situação, um valor de *hash* calculado sobre o criptograma poderia ser enviado por Ana junto com o criptograma e verificado por Beto. Deste modo, as corrupções acidentais ou maliciosas do criptograma poderiam ser detectadas por Beto.

Uma função de resumo criptográfico, também chamada de função de *hash* segura, gera uma sequência de bits, chamado de valor do *hash*, único para os dados de entrada da função. O *hash* é muito menor que o texto claro de entrada e geralmente tem um tamanho fixo de dezenas (algumas centenas) de bits. Em princípio, a função de *hash* é unidirecional porque não é reversível; isto é, não é computacionalmente possível recuperar o texto claro original a partir da sequência binária do *hash*. Além disso, idealmente, não é fácil achar dois conjuntos de dados (dois textos claros) que geram o mesmo valor de *hash*. A Figura 5.6 ilustra o comportamento das funções de *hash* seguras e também descreve algumas das propriedades técnicas destas funções que as tornam importantes para a segurança criptográfica.

As funções de resumo criptográfico não resolvem todos os problemas de integridade da mensagem. Por exemplo, mesmo os criptogramas acompanhados de valores *hash* ainda podem ser substituídos (incluindo a troca do *hash*) quando da transmissão de uma mensagem por um canal inseguro. Este problema é resolvido pelo uso de uma *tag* de autenticação no lugar do *hash*. As funções de resumo criptográfico podem ser usadas como blocos de construção das funções que calculam os códigos de autenticação de mensagens (*Message Authentication Codes* - MAC) baseados em *hash* e chamados de HMAC. Em termos simples, um HMAC é um *hash* chaveado e pode ser usado como um mecanismo de autenticação simples quando as partes comunicantes compartilham uma chave secreta. Neste caso, diz-se que a mensagem é autêntica para Ana e Beto apenas, mas não para um terceiro, uma vez que o terceiro não consegue determinar com certeza, quem (Ana ou Beto) gerou o MAC da mensagem.

Um código de autenticação de mensagem (MAC) é um mecanismo criptográfico que produz um selo (*tag*) de autenticidade baseado em uma chave compartilhada entre Ana e Beto e, por isto, relativa apenas a eles. A função MAC recebe como entrada a chave simétrica e

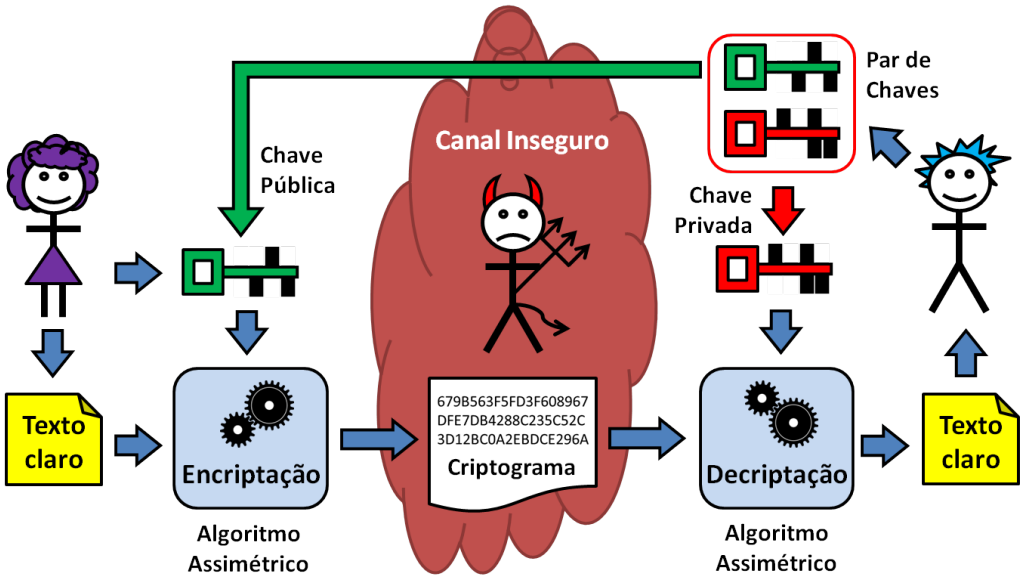


Figura 5.7. Sistema criptográfico assimétrico para sigilo (de [Braga and Dahab 2015]).

a mensagem (texto claro) e, utilizando em seu algoritmo funções de encriptação ou de *hash*, produz a *tag*. A verificação da *tag* é feita pela comparação da *tag* recebida com uma nova *tag* calculada no recebimento da mensagem.

A encriptação autenticada combina em uma única função criptográfica as funções de encriptação e de MAC. A encriptação autenticada é uma função criptográfica relativamente nova que atende aos dois objetivos (encriptação e autenticação) simultaneamente, sendo geralmente materializada por modos de operação específicos para algoritmos de encriptação de blocos, tais como os modos de operação GCM [NIST 2007] do AES e CCM, aplicável a qualquer encriptação de blocos.

5.3.3. Criptografia assimétrica (de chave pública)

Na criptografia de chave pública, uma das chaves do par é dita privada (a de decriptação), a outra é feita pública (a de encriptação). Estas duas chaves são matematicamente relacionadas e trabalham aos pares, de modo que o criptograma gerado com uma chave deve ser decriptado pela outra chave do par. Nenhuma das chaves do par pode ser usada sozinha em um sistema criptográfico assimétrico completo. A criptografia assimétrica pode ser usada tanto para encriptação (atuando na preservação de sigilo), quanto para assinaturas digitais (atuando na garantia de autenticidade e de irrefutabilidade). A criptografia de chave pública é indispensável para o sigilo e a privacidade na Internet, tornando possível a comunicação privada em redes públicas.

5.3.3.1. Encriptação assimétrica para sigilo

Na garantia de sigilo, a encriptação com a chave pública torna possível que qualquer um envie criptogramas (textos cifrados) para o dono da chave privada. A Figura 5.7 ilustra, com os mesmo personagens anteriores, um sistema criptográfico assimétrico para sigilo e seus elementos básicos. As mensagens de Ana para Beto são transmitidas por um canal inseguro, controlado

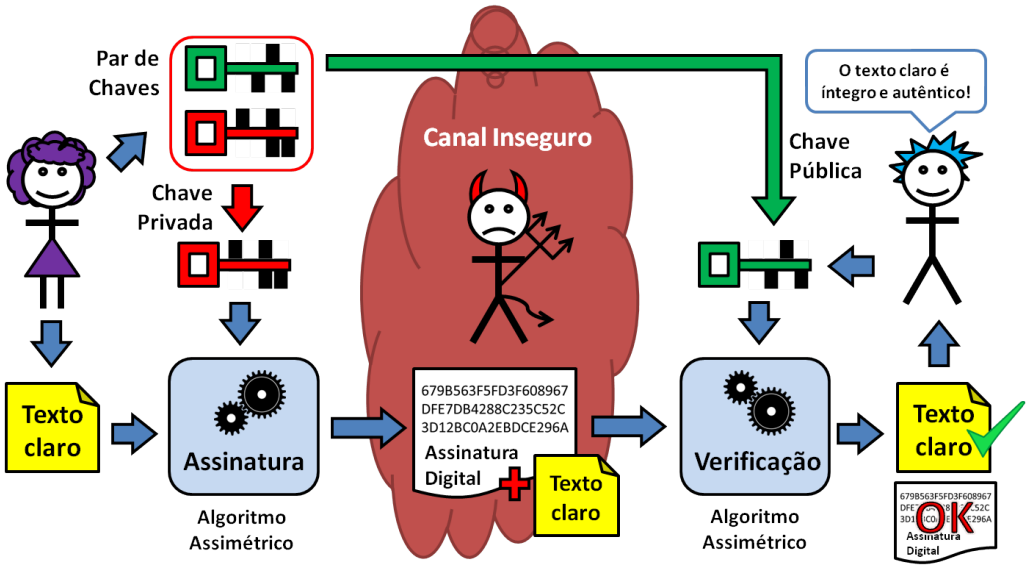


Figura 5.8. Criptografia assimétrica para autenticidade (de [Braga and Dahab 2015]).

por Ivo. Beto possui um par de chaves, uma chave pública e outra privada. Ana conhece a chave pública de Beto, mas somente o dono do par de chaves (Beto) conhece a chave privada.

Na Figura 5.7, observa-se que Ana envia uma mensagem privada para Beto. Para fazer isso, Ana encripta a mensagem usando a chave pública de Beto. Ana conhece a chave pública de Beto, que foi divulgada por ele previamente. Porém, o criptograma só pode ser decriptado pela chave privada de Beto; nem Ana pode fazê-lo. Para obter comunicação segura bidirecional, acrescenta-se ao sistema criptográfico o mesmo processo no sentido oposto (de Beto para Ana), com outro par de chaves. Isto é, Beto usa a chave pública de Ana para enviar mensagens encriptadas para ela. Já Ana usa sua própria chave privada pessoal para decriptar a mensagem enviada por Beto.

5.3.3.2. Criptografia assimétrica para autenticidade e irrefutabilidade

A criptografia de chave pública é a base para outros dois serviços criptográficos importantes para a proteção das comunicações: a autenticação das partes comunicantes e a verificação de integridade das mensagens. Os passos de operação da criptografia de chave pública para autenticação de mensagens, em certa medida, são o oposto do uso para sigilo. A criptografia de chave pública para assinatura digital é usada para obter integridade, autenticidade e irrefutabilidade.

Chama-se assinatura digital ao resultado de uma certa operação criptográfica com a chave privada sobre o texto claro. Neste caso, o dono da chave privada pode gerar mensagens assinadas, que podem ser verificadas por qualquer um que conheça a chave pública correspondente, portanto, sendo capaz de verificar a autenticidade da assinatura digital. Do ponto de vista das implementações criptográficas subjacentes, nem sempre a operação de assinatura é uma encriptação e nem a sua verificação é sempre uma decriptação.

Visto que qualquer um de posse da chave pública pode *"decriptar"* a assinatura digital, ela não é mais secreta, mas possui outra propriedade: a irrefutabilidade. Isto é, quem quer que

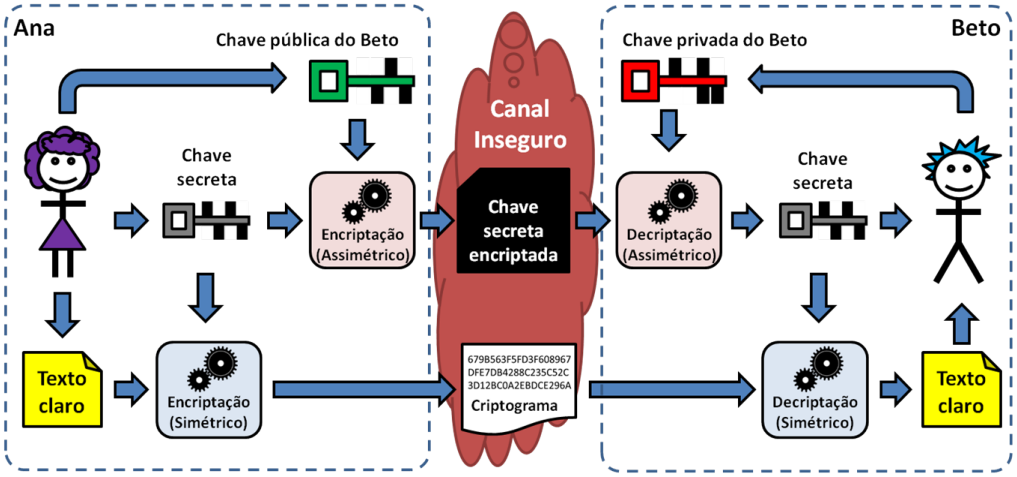


Figura 5.9. Transporte de chave de sessão (de [Braga and Dahab 2015]).

verifique a assinatura com a chave pública, sabe que ela foi produzida por uma chave privada exclusiva; logo, a mensagem não pode ter sido gerada por mais ninguém além do proprietário da chave privada. Na Figura 5.8, Ana usa sua chave privada para assinar digitalmente uma mensagem para Beto. O texto claro e a assinatura digital são enviados por um canal inseguro (sem sigilo) e podem ser lidos por todos, por isso a mensagem não é secreta. Qualquer um que conheça a chave pública de Ana (todo mundo, inclusive Beto), pode verificar a assinatura digital. Ivo pode ler a mensagem, mas não consegue falsificá-la de maneira indetectável, pois não conhece a chave privada de Ana.

5.3.4. Transporte de chaves de sessão e sistemas criptográficos híbridos

Há ocasiões em que entidades que nunca antes compartilharam chaves criptográficas precisam se comunicar em sigilo. Nestes casos, uma chave efêmera, usada apenas para algumas encriptações decorrentes de uma conversa, pode ser gerada por uma das partes e transportada com segurança para a outra parte, momentos antes do início da conversa. Tal transporte seguro de chaves pode ser implementado por um sistema criptográfico híbrido e emprega o conceito de chave de sessão, em que uma chave secreta é encriptada por uma chave pública para transporte seguro. Assim, a criptografia assimétrica (tipicamente, o RSA) é usada como canal seguro para o compartilhamento da uma chave secreta (por exemplo, do AES) usada na encriptação de dados da comunicação.

A Figura 5.9 mostra um sistema criptográfico híbrido usado para transporte de uma chave de sessão. Durante uma comunicação segura, a encriptação assimétrica só é executada uma única vez, no início da comunicação, para compartilhar a chave secreta. A partir do segundo criptograma da conversa, basta para Ana encriptar com a chave de sessão. A tarefa de Beto é recuperar a chave secreta e manter a comunicação com Ana usando a chave secreta de sessão para proteger os dados. De modo análogo, na decriptação, a recuperação da chave secreta só ocorre no início da comunicação. Observados os detalhes de implementação (por exemplo, derivando-se duas chaves de sessão a partir do segredo compartilhado), depois que ambos (Ana e Beto) possuem a chave secreta da sessão, a comunicação pode ocorrer nos dois sentidos. Isto é, tanto de Ana para Beto, quanto de Beto para Ana.

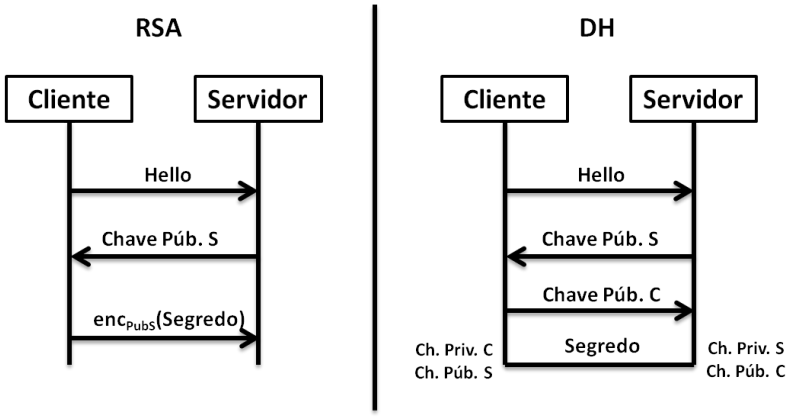


Figura 5.10. Transporte de chave com RSA e acordo de chave com DH.

5.3.5. RSA e Diffie-Hellman (DH)

Tradicionalmente, o algoritmo criptográfico assimétrico mais conhecido para encriptação e transporte de chaves é o RSA (publicado em 1978 [Rivest et al. 1978]), cujo nome é formado pelas letras iniciais dos sobrenomes dos seus autores: Ron Rivest, Adi Shamir e Leonard Adleman. Na prática, a versão acadêmica original (canônica) nunca deve ser utilizada. Implementações padronizadas que adotam mecanismos de preenchimento e aleatorização considerados seguros devem ser preferidas em vez das implementações fora dos padrões.

Para promover segurança e interoperabilidade, o uso e a implementação do RSA devem obedecer aos padrões internacionais. O documento PKCS#1v2 [Jonsson and Burt Kaliski 2003] especifica o *Optimal Asymmetric Encryption Padding* (OAEP) como um mecanismo de preenchimento (*padding*), que transforma o RSA em um mecanismo de encriptação assimétrica aleatorizado chamado RSA-OAEP. Já o *Probabilistic Signature Scheme* (PSS) é um esquema de assinaturas digitais probabilísticas com RSA (RSA-PSS) também padronizado no PKCS#1v2 [Jonsson and Burt Kaliski 2003] para substituir o esquema de assinatura do PKCS#1v1, considerado inseguro.

Outra maneira de resolver a questão do compartilhamento de chaves criptográficas entre entidade que nunca se encontraram, mas que precisam se comunicar em sigilo, é por meio dos protocolos de acordo de chaves. Os métodos de acordo de chaves são utilizados para combinar ou negociar uma chave secreta entre dois ou mais participantes usando um canal público. Uma característica interessante destes métodos é que o segredo compartilhado (a partir do qual a chave será derivada) é combinado pela troca de informações públicas por meio de um canal inseguro.

O protocolo de acordo de chaves Diffie-Hellman (DH), publicado em 1976 no artigo que lançou a criptografia de chave pública [Diffie and Hellman 1976], é o método mais usado na Internet para o acordo de um segredo compartilhado entre duas partes. Assim como no caso do RSA, as implementações canônicas do DH não devem ser usadas por que elas são vulneráveis aos ataques de personificação de uma ou ambas as partes (o ataque *Man-In-The-Middle* - MITM). Duas variações do DH original são aquela com parâmetros efêmeros (DHE) e aquela com autenticação das partes (DHA). Também há variações do DH implementadas como a criptografia de curvas elípticas. Figura 5.10 ilustra as diferenças entre as duas abordagens de distribuição de chaves, RSA e DH, quando usadas em protocolos como o SSL/TLS.

5.3.6. Criptografia de Curvas Elípticas

Os sistemas criptográficos baseados em curvas elípticas foram propostos por volta de 1985 de forma independente por dois pesquisadores [Miller 1985, Koblitz 1987], e se baseiam na dificuldade de se calcular o logaritmo discreto no grupo de pontos induzido por uma curva elíptica definida sobre um corpo finito F_{p^m}, onde p é um número primo e m é um inteiro positivo. Neste texto nos limitamos aos seguintes casos: $p \neq 2, m = 1$, isto é, corpos *primos*; e $p = 2, m \geq 1$, chamados de corpos *binários*. A forma geral da equação de uma curva elíptica é $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$ [Hankerson et al. 2004]. Para corpos primos simplifica-se a forma da curva: $y^2 = x^3 + ax + b$, com a,b em F_p. Para corpos binários, a forma da curva é $y^2 + xy = x^3 + ax^2 + b$, com a,b em F_{2^m}. O tamanho das chaves criptográficas nos criptossistemas de curvas elípticas é consideravelmente menor que nos sistemas criptográficos assimétricos tradicionais, como o RSA ou ElGamal. A criptografia de curvas elípticas é usada em trocas de chaves, assinaturas digitais e encriptação. Os esquemas e protocolos criptográficos sobre curvas elípticas mais comumente utilizados atualmente são o protocolo de acordo de chaves *Elliptic Curve Diffie Hellman* (ECDH), o esquema de assinaturas digitais *Elliptic Curve Digital Signature Algorithm* (ECDSA) [NIST 2013] e a encriptação *Elliptic Curve Integrated Encryption Scheme* (ECIES) [Hankerson et al. 2004].

O NIST [NIST 2013] recomenda cinco curvas elípticas sobre corpos primos para serem utilizadas no ECDSA, em cinco níveis de segurança, a saber: P-192, P-224, P-256, P-384 e P-521 (também conhecidas [SEC 2010] como secp192r1, secp224r1, secp256r1, secp384r1, secp521r1). Há uma variedade de curvas definidas por diversas instituições, muitas já consideradas inseguras para os padrões de segurança atuais. Há também padrões emergentes, como a curva 25519 [Bernstein 2006], que tem sido considerada um novo padrão de fato na indústria de segurança criptográfica e uma substituta para as curvas padronizadas pelo NIST [NIST 2013]. Esta curva é mais rápida, possivelmente mais segura, e alegadamente sem a influência de agências de governo sobre seu projeto (conforme `https://cr.yp.to/ecdh.html`).

5.3.7. Tamanhos de chaves criptográficas e níveis de segurança

Sistemas criptográficos implementados em software geralmente combinam diversas funções criptográficas, cujas configurações devem ser escolhidas de modo a manter as funções no mesmo nível de segurança. A Tabela 5.1 (apresentada em [Braga and Dahab 2018]) mostra uma comparação entre os níveis de segurança de tipos de primitivas criptográficas e os tamanhos de chaves correspondentes. O nível de segurança é uma medida relativa e pode ser interpretado como sendo a força de um algoritmo criptográfico simétrico com chave de n bits.

A tabela foi adaptada de `keylength.com` e considera as recomendações do NIST (2016) [BlueKrypt]. Cada linha da tabela representa um nível de segurança, de 80 bits até 256 bits. Por exemplo, no nível de 80 bits, atualmente útil apenas para sistemas legados, chaves RSA de 1024 bits são comparáveis às chaves DH de 1024 bits com segredo de 160 bits, uma curva elíptica de 160 bits e *hashes* de 160 bits também. Já no nível de segurança mais alto, 256 bits, o RSA se torna inviável na prática com chaves de 15360 bits, enquanto as curvas elípticas despontam com chaves de 512 bits e *hashes* de 512 bits para assinaturas e de até 384 bits para geração de chaves.

Tabela 5.1. Níveis de segurança e tamanhos de chave (de [BlueKrypt]).

Nível de segurança	Fatoração (IFP)	DLP		Curva Elíptica	Hash	
		chave	Corpo		Assinatura	KDF/HMAC/PRNG
80	1024	160	1024	160–163	SHA1 (160)	–
112	2048	224	2048	224–233	SHA-224/512 SHA3-224	–
128	3072	256	3072	256–283	SHA-256/512 SHA3-256	SHA1(160)
192	7680	384	7680	384–409	SHA-384 SHA3-384	SHA-224 SHA-512
256	15360	512	15360	512–571	SHA-512 SHA3-512	SHA-256/384/512 SHA-512/SHA3-512

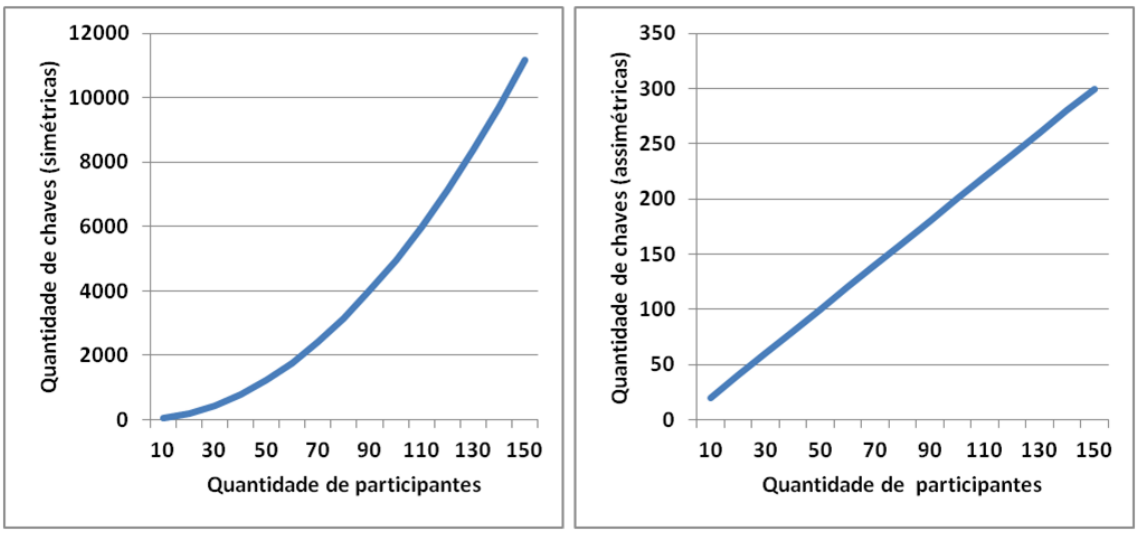


Figura 5.11. Quantidades de chaves assimétricas e simétricas para 150 participantes (de [Braga and Dahab 2018]).

5.3.8. Distribuição de chaves públicas com certificação digital

A criptografia assimétrica (de chaves públicas) tem a propriedade de simplificar o problema de distribuição de chaves criptográficas. Por exemplo, cada um dos participantes do sistema criptográfico só precisa ter o seu par de chaves, manter em segredo a sua chave privada e divulgar a chave pública. A Figura 5.11 ilustra a distribuição de chaves públicas para um grupo de quatro indivíduos. O gráfico da direita na Figura 5.11 mostra o crescimento da quantidade de chaves (públicas e privadas) como uma função afim da quantidade de participantes, onde quantidade de chaves é o dobro da quantidade de participantes. Já a distribuição de chaves simétricas (gráfico da esquerda) obedece a uma função quadrática.

O principal problema administrativo associado à distribuição de chaves públicas é justamente a confiança depositada na chave distribuída publicamente. Se a chave pública de alguém pode ser facilmente encontrada em qualquer lugar, então é difícil assegurar com alto grau de certeza que esta chave não foi corrompida ou substituída. O problema de garantir a integridade e a autenticidade da chave pública é muito importante e, se não for solucionado satisfatoriamente, pode comprometer a confiança no sistema criptográfico inteiro. Uma maneira de validar chaves públicas é fazer com que elas sejam emitidas por Autoridades Certificadoras

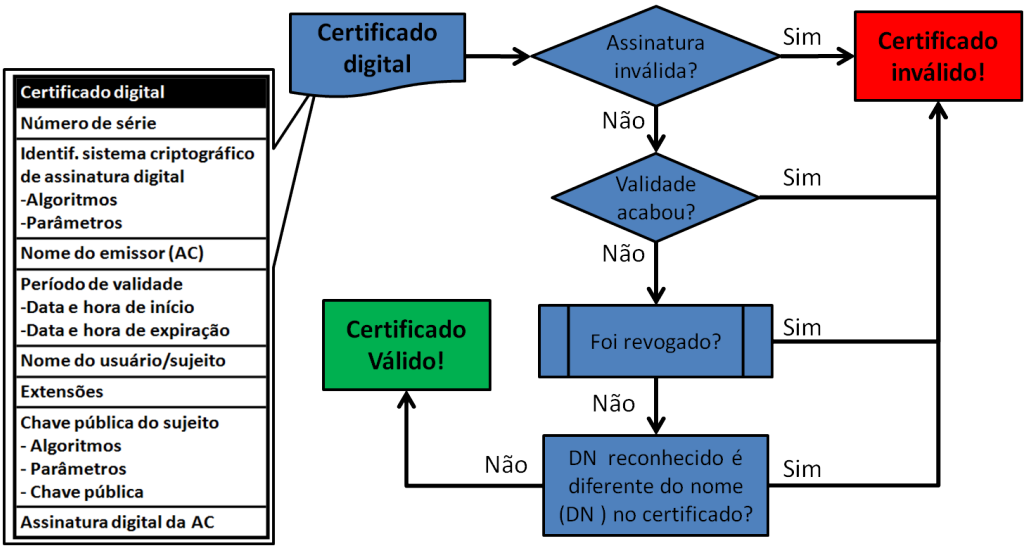


Figura 5.12. Fluxograma de validação de um certificado digital (de [Braga and Dahab 2015]).

(AC) de uma Infraestrutura de Chaves Públicas (ICP), do inglês *Public-Key Infrastrucuture* - PKI, que torne possível a verificação da autenticidade de tais chaves.

Certificação digital e os aspectos de validação de certificados digitais possuem atualmente boas práticas bem documentadas [NIST 2012] e literatura especializada na implantação de tais infraestruturas [Chandra et al. 2002]. Um certificado digital de chave pública, ou simplesmente certificado, é uma estrutura de dados que dá como verdadeiro o vínculo entre uma chave pública autêntica e uma entidade cujo nome está no certificado. A veracidade do certificado é garantida por uma terceira parte confiável, emissora do certificado, chamada de Autoridade Certificadora (CA). O certificado contém a assinatura digital da CA emissora, a chave pública e a identidade da entidade reconhecida pela CA, além de outras informações, tais como as datas de início e de final de validade e o uso pretendido da chave, entre outras. Por isto, o certificado é usado na verificação de que uma chave pública válida pertence a uma entidade e serve também como meio confiável para distribuição de chaves públicas.

A validação do certificado é realizada toda vez que a confiança na autenticidade da chave pública contida nele deve ser garantida. A assinatura da AC pode ser verificada por qualquer um com acesso à chave pública da AC, cujo certificado é amplamente disponível. Por exemplo, num caso bastante comum, uma AC pode emitir certificados SSL/TLS para servidores web que se comunicam por meio do protocolo HTTP sobre TLS (HTTPS). Quando um software cliente HTTPS (o navegador ou *browser*) faz uma requisição SSL/TLS de saudação para um servidor web protegido, em que a resposta do servidor inclui o seu certificado digital. O software cliente HTTPS valida o certificado do servidor verificando a assinatura da AC emissora sobre a chave pública do servidor e outros parâmetros do certificado. Se o cliente já não possuir a chave pública da AC, ele vai buscá-la (em um repositório de chaves públicas da AC). Se o certificado é verificado como válido, então o software cliente acredita que o servidor é autêntico.

A validação do certificado (e da chave pública contida nele) abrange mais etapas do que somente a verificação da assinatura da AC. Além da verificação da assinatura, o software cliente precisa verificar se o certificado não atingiu o final de seu período de validade, se não foi

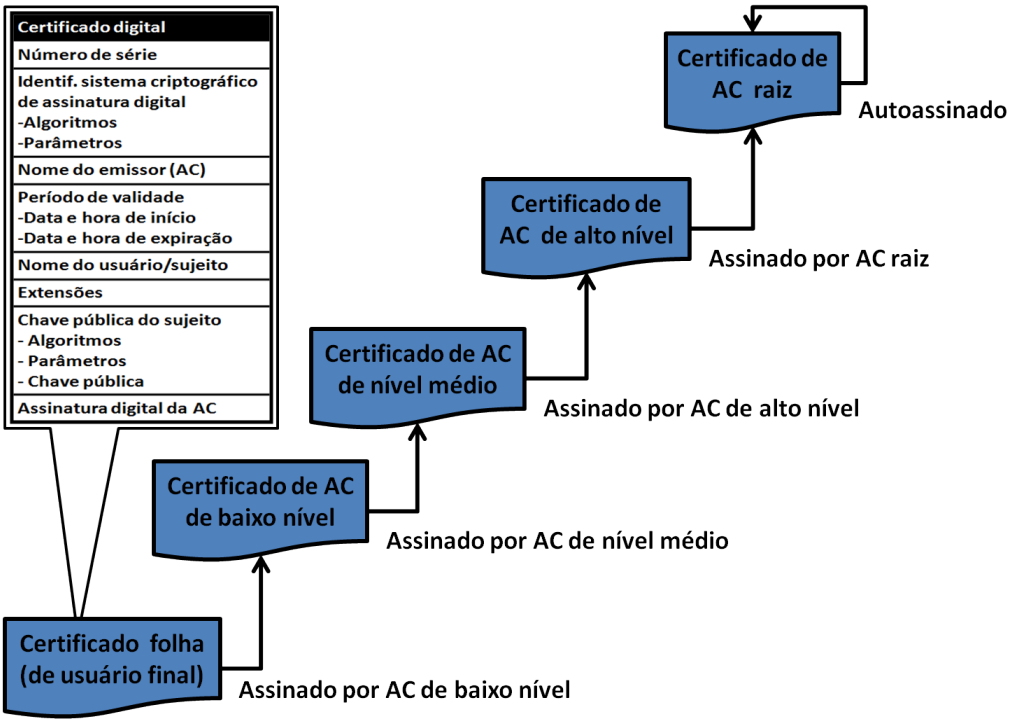


Figura 5.13. Cadeia hierárquica de certificação digital (de [Braga and Dahab 2015]).

revogado e se o nome constante no certificado é o mesmo da entidade que alega ser a dona da chave pública. O nome também deve ser obtido de um terceiro confiável, por exemplo, de um serviço de DNS seguro, no caso de nomes de domínio. A validação do certificado é ilustrada no fluxograma da Figura 5.12.

A verificação da assinatura da AC em um certificado digital exige a chave pública da AC. Para ser confiável, a chave pública da AC deve estar contida em um certificado assinado por outra AC ou autoassinado, se for uma CA raiz. As verificações sucessivas de uma sequência de assinaturas formam uma cadeia de certificação ou hierarquia de certificados. Os certificados na base da hierarquia são assinados pelas ACs de mais baixo nível, cujos certificados são assinados pelas ACs intermediárias, que têm seus certificados assinados pelas ACs de alto nível, cujos certificados são assinados pela AC raiz, que tem seus certificados autoassinados. A Figura 5.13 ilustra esta cadeia de certificação.

Uma AC revoga um certificado nas seguintes situações: (*i*) quando ocorrem erros na emissão do certificado (por exemplo, quando o nome da entidade está escrito de forma errada); (*ii*) o certificado foi emitido para uso de um serviço específico que o portador não tem mais acesso (por exemplo, no caso de demissão de um funcionário); (*iii*) a chave privada do usuário final é comprometida (por exemplo, um *smartcard* foi perdido); ou ainda (*iv*) quando a chave privada da AC é comprometida (por exemplo, em uma situação extrema, quando a credencial de um administrador da AC é comprometida). Uma Lista de Certificados Revogados (*Certificate Revocation List* - CRL) é o documento assinado digitalmente pela AC que lista o número de série de todos os certificados, ainda não expirados, que perderam a utilidade por algum dos motivos acima. Um software criptográfico pode consultar um serviço de CRL (oferecido pela AC) para receber atualizações periódicas, em intervalos regulares definidos por procedimentos da AC.

5.4. Criptografia aplicada com `OpenSSL`

Além de ser uma ferramenta para utilização e testes do protocolo SSL/TLS, o software `OpenSSL` [OpenSSL.org] também é um biblioteca criptográfica completa com muitas funções de criptografia disponíveis por meio da sua interface de comandos de linha (*Command Line Interface* - CLI). Por isto, esta seção usa a CLI do `OpenSSL` para mostrar na prática os conceitos descritos na seção anterior. O `OpenSSL` é amplamente disponível, já que, por exemplo, faz parte das distribuições Linux. Já os usuários Windows devem baixar os binários do `OpenSSL` de uma fonte confiável, facilmente encontrada. Em qualquer caso, é importante verificar qual versão do `OpenSSL` está em uso.

Os exemplos mostrados a seguir neste capítulo foram executados com o `OpenSSL` para Windows em uma janela de terminal (do tipo DOS) e funcionam da mesma forma na versão para Linux. O leitor interessado em reproduzir as tarefas deste tutorial, quando pertinente, pode executar os comandos *OpenSSL* na sequência em que são apresentados nas listagens a seguir.

5.4.1. Identificação da versão instalada do `OpenSSL`

A Listagem 5.1 mostra dois modos de verificar a versão do `OpenSSL` instalado. A primeira é com o comando `openssl version`, da linha 1, que mostra o número de versão e a data de lançamento da versão. A segunda é com a opção `version -a` (linha 4) que lista todas as informações detalhadas sobre a versão. Como pode ser observado na Listagem 5.1 (linhas de 5 até 7), a versão 1.1.1 de agosto de 2018, para Windows de 32 bits, foi usada durante a escrita deste texto. Esta distribuição do `OpenSSL` já suporta a versão 1.3 do SSL/TLS.

Nas linhas de 7 até 17, a Listagem 5.1 mostra diversas informações relacionadas ao binário da distribuição do `OpenSSL` e seu processo de compilação e *build*. Estes são detalhes de implementação que não afetam o tutorial desta seção.

A Listagem 5.1 serve de modelo para as outras listagens mostradas ao longo do restante do texto. Por isto, o leitor deve se habituar a este formato de listagem de comandos em que uma sequência de comandos e suas saídas são mostradas na mesma listagem de console (janela de terminal) do sistema operacional.

Listagem 5.1. Descobrindo a versão do `OpenSSL`.

```
1 C:\>openssl version
2 OpenSSL 1.1.1  11 Sep 2018
3
4 C:\>openssl version -a
5 OpenSSL 1.1.1  11 Sep 2018
6 built on: Thu Sep 13 14:54:36 2018 UTC
7 platform: VC-WIN32
8 options:  bn(64,32) rc4(8x,mmx) des(long) idea(int) blowfish(ptr)
9 compiler: cl /Z7 /Fdossl_static.pdb /Gs0 /GF /Gy /MD /W3 /wd4090 /nologo /O2 /WX
10 -DL_ENDIAN -DOPENSSL_PIC -DOPENSSL_CPUID_OBJ -DOPENSSL_BN_ASM_PART_WORDS -DOPEN
11 SSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_AS
12 M -DSHA512_ASM -DRC4_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM -DVPAES_ASM -DWHIRLPOO
13 L_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DPADLOCK_ASM -DPOLY1305_ASM -D_USE_32BIT_T
14 IME_T -D_USING_V110_SDK71_ -D_WINSOCK_DEPRECATED_NO_WARNINGS
15 OPENSSLDIR: "C:\Program Files (x86)\Common Files\SSL"
16 ENGINESDIR: "C:\Program Files (x86)\OpenSSL\lib\engines-1_1"
17 Seeding source: os-specific
```

5.4.2. Os comandos principais da CLI de serviços criptográficos

O `OpenSSL` é uma ferramenta bastante flexível que pode, inclusive, ser utilizado como um provedor de serviços criptográficos com uma interface de linha da comando. A Listagem 5.2 contém a saída do comando `openssl help` que mostra os comandos padrão (nas linhas de 2 a 14), com enfase em dois comandos em particular.

Primeiro é o comando `dgst`, que é usado para cálculo de *hashes*, MACs e assinaturas digitais. O segundo é o comando `enc`, usado para encriptação simétrica. A Listagem 5.2, nas linhas de 16 até 22, mostra todas as funções de *hash* disponíveis na instalação do `OpenSSL`.

Em seguida, as linhas de 24 até 45 mostram as funções de encriptação simétrica no formato AAA-KKK-OOO, onde AAA é o algoritmo de encriptação, KKK é o tamanho de chave criptográfica e OOO é o modo de operação do algoritmo de encriptação simétrico. Cada comando tem seu próprio `help`, que deve ser explorado pelo leitor.

Listagem 5.2. Descobrindo os comandos do `OpenSSL`.

```
1 C:\>openssl help
2 Standard commands
3 asn1parse         ca              ciphers         cms
4 crl               crl2pkcs7       dgst            dhparam
5 dsa               dsaparam        ec              ecparam
6 enc               engine          errstr          gendsa
7 genpkey           genrsa          help            list
8 nseq              ocsp            passwd          pkcs12
9 pkcs7             pkcs8           pkey            pkeyparam
10 pkeyutl          prime           rand            rehash
11 req              rsa             rsautl          s_client
12 s_server         s_time          sess_id         smime
13 speed            spkac           srp             storeutl
14 ts               verify          version         x509
15
16 Message Digest commands (see the 'dgst' command for more details)
17 blake2b512       blake2s256      gost            md4
18 md5              mdc2            rmd160          sha1
19 sha224           sha256          sha3-224        sha3-256
20 sha3-384         sha3-512        sha384          sha512
21 sha512-224       sha512-256      shake128        shake256
22 sm3
23
24 Cipher commands (see the 'enc' command for more details)
25 aes-128-cbc      aes-128-ecb     aes-192-cbc     aes-192-ecb
26 aes-256-cbc      aes-256-ecb     aria-128-cbc    aria-128-cfb
27 aria-128-cfb1    aria-128-cfb8   aria-128-ctr    aria-128-ecb
28 aria-128-ofb     aria-192-cbc    aria-192-cfb    aria-192-cfb1
29 aria-192-cfb8    aria-192-ctr    aria-192-ecb    aria-192-ofb
30 aria-256-cbc     aria-256-cfb    aria-256-cfb1   aria-256-cfb8
31 aria-256-ctr     aria-256-ecb    aria-256-ofb    base64
32 bf               bf-cbc          bf-cfb          bf-ecb
33 bf-ofb           camellia-128-cbc camellia-128-ecb camellia-192-cbc
34 camellia-192-ecb camellia-256-cbc camellia-256-ecb cast
35 cast-cbc         cast5-cbc       cast5-cfb       cast5-ecb
36 cast5-ofb        des             des-cbc         des-cfb
37 des-ecb          des-ede         des-ede-cbc     des-ede-cfb
38 des-ede-ofb      des-ede3        des-ede3-cbc    des-ede3-cfb
39 des-ede3-ofb     des-ofb         des3            desx
40 idea             idea-cbc        idea-cfb        idea-ecb
41 idea-ofb         rc2             rc2-40-cbc      rc2-64-cbc
42 rc2-cbc          rc2-cfb         rc2-ecb         rc2-ofb
43 rc4              rc4-40          seed            seed-cbc
44 seed-cfb         seed-ecb        seed-ofb        sm4-cbc
45 sm4-cfb          sm4-ctr         sm4-ecb         sm4-ofb
```

5.4.3. Codificação Base64 e geração de números pseudo-aleatórios

Algoritmos criptográficos produzem resultados em sequências binárias que, muitas vezes, são transmitidos ou armazenados em formato texto. Para isto, o `OpenSSL` oferece o comando `enc -base64` para codificação de um arquivo de entrada qualquer no formato Base64, aplicando uma regra simples: cada 3 bytes (24 bits) binários viram 4 caracteres imprimíveis (cada um com 6 bits e completados para um byte). A codificação Base64 não é uma função criptográfica e não tem finalidade de segurança. A Listagem 5.3 mostra como codificar e decodificar no formato Base64 um arquivo `gabarito.txt`. Este arquivo é usado em vários exemplos neste capítulo e (como seu nome sugere) simula o gabarito de uma prova, uma informação que certamente deve ser protegida de várias formas diferentes.

Listagem 5.3. Codificação em Base64 com `OpenSSL`.

```
1 C:\>type gabarito.txt
2 Gabarito da prova
3 1-a 2-c 3-d 4-c 5-a 6-b 7-d 8-d 9-a 10-c
4
5 C:\>openssl enc -base64 -in gabarito.txt -out gabarito.b64
6
7 C:\>type gabarito.b64
8 R2FiYXJpdG8gZGEgcHJvdmENCjEtYSAyLWMgMy1kIDQtYyA1LWEgNi1iIDctZCA4LWQgOS1hIDEwLWMNCg==
9
10 C:\>openssl enc -d -base64 -in gabarito.b64 -out gabarito2.txt
11
12 C:\>type gabarito2.txt
13 Gabarito da prova
14 1-a 2-c 3-d 4-c 5-a 6-b 7-d 8-d 9-a 10-c
```

Outro comando do `OpenSSL` bastante útil é o `rand` usado na geração de sequências de bytes pseudo-aleatórias. A Listagem 5.4 mostra diversas maneiras de utilizar o comando `rand`. A linha 1 mostra como gerar uma sequência pseudo-aleatória em hexadecimal de 20 bytes. As linhas 4 e 7 geram sequências pseudo-aleatórias de 32 e 64 bytes, respectivamente. A linha 11 formata a sequência de saída em base64. A linha 14 escreve a sequência de saída em um arquivo com a opção `-rand`. Isto é útil para fornecer uma fonte de aleatoriedade para outras funções criptográficas, como por exemplo a geração de chaves. A linha 16 mostra o conteúdo do arquivo. A linha 19 grava o arquivo e mostra a saída no formato Base64.

Listagem 5.4. Geração de números pseudo-aleatórios.

```
1 C:\>openssl rand -hex 20
2 031954f853219d94295a4dad7feefd85bbd5096e
3
4 C:\>openssl rand -hex 32
5 9047b1cc5050be773ed87c6d670f3a5ee1438a114b198aad0b6767969dfdab4b
6
7 C:\>openssl rand -hex 64
8 c9157b55870206b2acfc9149530ef6f5a9edda45e9970e88b68e153446f873062dd56bd9318e9c1e
9 bc5410a873fc7e7fba7ff9d45b4f907d2f8719d1ab7d0f54
10
11 C:\>openssl rand -base64 64
12 ruZqMy+qjAdmQgWP2bhpNoXSwhJsB7WrY5IAiPzMpXsG9oYRMuELBccoLVGg1clyU4BTR4AfzLxqMi+00MHbog==
13
14 C:\>openssl rand -base64 -out random.txt 64
15
16 C:\>type random.txt
17 E1fK2vu0Y5h8xPtJRruUUvUO96XYi9KLYOlpf7mwE6pUW3gygmtq+UZxRpaS701+O86235HIYCaqZ8jk4b/UYQ==
18
19 C:\>openssl rand -base64 -rand random.txt 64
20 r2byK4Z+5GdfH+1S3V2OWRCSLklNX2g4v0YH/ms7FR2zVhbykuZY6RvHRtXjpDSeccle1Udz6tFnyVJh2PkcNg==
```

5.4.4. Encriptação simétrica e os algoritmos disponíveis

O `OpenSSL` oferece uma variedade de funções de encriptação simétrica. A Listagem 5.5 mostra a saída do comando `enc -ciphers`, em que se pode observar grupos de funções bem definidos, com algoritmos criptográficos, tamanhos de chaves e modos de operação. Em particular, observam-se os seguintes algoritmos criptográficos: AES, ARIA, blowfish (bf), camelia, cast, chacha20, DES, IDEA, RC4, entre outros. Todas estas funções de encriptação simétrica estão disponíveis pela CLI do `OpenSSL`. As próximas seções mostram usos de algumas delas.

Listagem 5.5. Lista dos algoritmos simétricos de encriptação do `OpenSSL`.

```
C:\>openssl enc -ciphers
Supported ciphers:
-aes-128-cbc               -aes-128-cfb               -aes-128-cfb1
-aes-128-cfb8              -aes-128-ctr               -aes-128-ecb
-aes-128-ofb               -aes-192-cbc               -aes-192-cfb
-aes-192-cfb1              -aes-192-cfb8              -aes-192-ctr
-aes-192-ecb               -aes-192-ofb               -aes-256-cbc
-aes-256-cfb               -aes-256-cfb1              -aes-256-cfb8
-aes-256-ctr               -aes-256-ecb               -aes-256-ofb
-aes128                    -aes128-wrap               -aes192
-aes192-wrap               -aes256                    -aes256-wrap
-aria-128-cbc              -aria-128-cfb              -aria-128-cfb1
-aria-128-cfb8             -aria-128-ctr              -aria-128-ecb
-aria-128-ofb              -aria-192-cbc              -aria-192-cfb
-aria-192-cfb1             -aria-192-cfb8             -aria-192-ctr
-aria-192-ecb              -aria-192-ofb              -aria-256-cbc
-aria-256-cfb              -aria-256-cfb1             -aria-256-cfb8
-aria-256-ctr              -aria-256-ecb              -aria-256-ofb
-aria128                   -aria192                   -aria256
-bf                        -bf-cbc                    -bf-cfb
-bf-ecb                    -bf-ofb                    -blowfish
-camellia-128-cbc          -camellia-128-cfb          -camellia-128-cfb1
-camellia-128-cfb8         -camellia-128-ctr          -camellia-128-ecb
-camellia-128-ofb          -camellia-192-cbc          -camellia-192-cfb
-camellia-192-cfb1         -camellia-192-cfb8         -camellia-192-ctr
-camellia-192-ecb          -camellia-192-ofb          -camellia-256-cbc
-camellia-256-cfb          -camellia-256-cfb1         -camellia-256-cfb8
-camellia-256-ctr          -camellia-256-ecb          -camellia-256-ofb
-camellia128               -camellia192               -camellia256
-cast                      -cast-cbc                  -cast5-cbc
-cast5-cfb                 -cast5-ecb                 -cast5-ofb
-chacha20                  -des                       -des-cbc
-des-cfb                   -des-cfb1                  -des-cfb8
-des-ecb                   -des-ede                   -des-ede-cbc
-des-ede-cfb               -des-ede-ecb               -des-ede-ofb
-des-ede3                  -des-ede3-cbc              -des-ede3-cfb
-des-ede3-cfb1             -des-ede3-cfb8             -des-ede3-ecb
-des-ede3-ofb              -des-ofb                   -des3
-des3-wrap                 -desx                      -desx-cbc
-id-aes128-wrap            -id-aes128-wrap-pad        -id-aes192-wrap
-id-aes192-wrap-pad        -id-aes256-wrap            -id-aes256-wrap-pad
-id-smime-alg-CMS3DESwrap  -idea                      -idea-cbc
-idea-cfb                  -idea-ecb                  -idea-ofb
-rc2                       -rc2-128                   -rc2-40
-rc2-40-cbc                -rc2-64                    -rc2-64-cbc
-rc2-cbc                   -rc2-cfb                   -rc2-ecb
-rc2-ofb                   -rc4                       -rc4-40
-seed                      -seed-cbc                  -seed-cfb
-seed-ecb                  -seed-ofb                  -sm4
-sm4-cbc                   -sm4-cfb                   -sm4-ctr
-sm4-ecb                   -sm4-ofb
```

5.4.5. Encriptação e decriptação com AES

A Listagem 5.6 mostra várias maneiras de encriptar e decriptar o arquivo de exemplo `gabarito.txt` com o comando `enc` e o algoritmo AES. O comando da linha 1 encripta (opção `-c`), codifica o criptograma de saída em Base64 (opção `-a`) e mostra os parâmetros (opção `-p`) da encriptação com AES em modo CTR e chave de 256 bits (`-aes-256-ctr`). Ainda neste comando, a chave é gerada a partir de uma senha (opção `-k`) com o método PBKDF2 (opção `-pbkdf2`). Os parâmetros mostrados em hexadecimal são o `salt` usado no PBKDF2, a chave criptográfica (`key`) e o vetor de inicialização (`iv`) usado como contador do modo CTR. O criptograma de saída é gravado no arquivo `gabarito.aes`. Se a senha não é passada com a opção (`-k`), o `OpenSSL` pede que uma senha seja digitada.

O comando `enc -d`, das linhas 6 e 7, faz a decriptação do criptograma contido no arquivo `gabarito.aes`, nas mesmas configurações utilizadas pela encriptação, gerando o arquivo de saída `gabarito4.txt`, cujo conteúdo é mostrado pelo comando `type` na linha 9 e consiste do texto claro recuperado.

O comando `enc -c`, das linhas de 14 até 16, não utiliza mais as opções `-pbkdf2` e `-k` (em minúsculo) para geração de chaves a partir de senhas. Em vez disto, a chave criptográfica é passada explicitamente no comando com a opção `-K` (em maiúsculo) seguida pela chave em hexadecimal (linha 15). O mesmo acontece com o IV passado com a opção `-iv` (linha 16). A opção `-salt` não é mais necessária uma vez que não há mais geração de parâmetros aleatórios internos a função. Vale lembrar que esta chave criptográfica pode ter sido gerada, por exemplo, pelo comando `rand` explicado anteriormente.

O comando `enc -d`, das linhas de 18 até 20, faz a decriptação do arquivo `gabarito.aes` nas mesmas configurações utilizadas na encriptação anterior, gerando o arquivo de saída `gabarito5.txt`, cujo conteúdo é mostrado pelo comando na linha 22.

Listagem 5.6. Encriptação com AES.

```
1 C:\>openssl enc -e -a -p -aes-256-ctr -pbkdf2 -in gabarito.txt -out gabarito.aes -k 1234
2 salt=BEF1B00E6BCC6C01
3 key=6CB40CAEF6FDEDB31DBD908256F63276DB8FBC1E3430CEF18B6E0F5CD112D5F2
4 iv =E570548E2B1376147E3342F08082E29A
5
6 C:\>openssl enc -d -a -salt -aes-256-ctr -pbkdf2 -in gabarito.aes -out gabarito4.txt
7  -k 1234
8
9 C:\>type gabarito4.txt
10 Gabarito da prova
11 1-a 2-c 3-d 4-c 5-a 6-b 7-d 8-d 9-a 10-c
12
13 C:\>openssl enc -e -a -aes-256-ctr -in gabarito.txt -out gabarito.aes
14  -K 6CB40CAEF6FDEDB31DBD908256F63276DB8FBC1E3430CEF18B6E0F5CD112D5F2
15  -iv E570548E2B1376147E3342F08082E29A
16
17 C:\>openssl enc -d -a -aes-256-ctr -in gabarito.aes -out gabarito5.txt
18  -K 6CB40CAEF6FDEDB31DBD908256F63276DB8FBC1E3430CEF18B6E0F5CD112D5F2
19  -iv E570548E2B1376147E3342F08082E29A
20
21 C:\>type gabarito5.txt
22 Gabarito da prova
23 1-a 2-c 3-d 4-c 5-a 6-b 7-d 8-d 9-a 10-c
```

5.4.6. Preenchimento de blocos na criptografia simétrica

Na encriptação de blocos, o texto claro deve ser um múltiplo do tamanho do bloco do algoritmo criptográfico. Quando o comprimento do texto claro não é múltiplo do tamanho do bloco, um método de (*padding*) é usado para preencher o último bloco incompleto do texto claro. A Listagem 5.7 mostra a encriptação sem *padding* e a opção `-nopad`. As linhas de 1 até 7 listam dois arquivos com bloco incompleto, um com 1 byte e outro com 15 bytes, mais um arquivo de bloco completo (16 bytes). O comando da linha 8 encripta com AES-128-ECB e `-nopad` um texto claro múltiplo do bloco. A decriptação está na linha 9. Todos os comandos usam a mesma chave.

Os comandos das linhas 12 e 13 encriptam com *padding* e decriptam sem *padding* um arquivo de 1 byte, enquanto os comandos das linhas 15 e 16 fazem o mesmo para um arquivo de 15 bytes. O resultado nos dois casos é um arquivo decriptado maior que o original e com tamanho de um bloco. Os bytes extras são o *padding*, mostrados na Figura 5.14, tela do software HxD (`https://mh-nexus.de/en/hxd`). Já os comandos das linhas 18 e 19 encriptam com *padding* e decriptam sem *padding* um arquivo de 16 bytes (um bloco) e o resultado da decriptação é um arquivo com dois blocos, o bloco extra é o *padding* (vide Figura 5.14).

Listagem 5.7. Prenchimento de blocos do AES.

```
1 C:\>type i1.txt
2 1
3 C:\>type iF.txt
4 FFFFFFFFFFFFFFF
5 C:\>type i16F.txt
6 0123456789ABCDEF
7
8 C:\>openssl enc -
      e -AES-128-ECB -in i16F.txt -K 6CB40CAEF6FDEDB31DBD908256F6327A -nopad -out o16F.txt
9 C:\>openssl enc -d -AES-128-ECB -in o16F.txt -K 6CB4...327A -nopad
10 0123456789ABCDEF
11
12 C:\>openssl enc -e -AES-128-ECB -in i1.txt -K 6CB4...327A -out o1.txt
13 C:\>openssl enc -d -AES-128-ECB -in o1.txt -K 6CB4...327A -out o1_2.txt -nopad
14
15 C:\>openssl enc -e -AES-128-ECB -in iF.txt -K 6CB4...327A -out oF.txt
16 C:\>openssl enc -d -AES-128-ECB -in oF.txt -K 6CB4...327A -out oF_2.txt -nopad
17
18 C:\>openssl enc -e -AES-128-ECB -in i16F.txt -K 6CB4...327A -out o16F.txt
19 C:\>openssl enc -d -AES-128-ECB -in o1F.txt -K 6CB4...327A -out o16F_2.txt -nopad
```

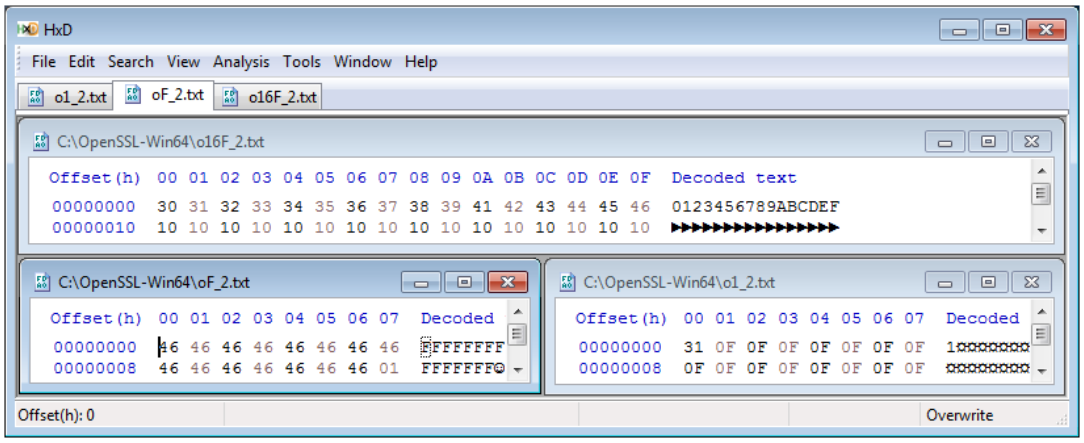


Figura 5.14. Preenchimentos dos blocos parciais e do bloco completo.

5.4.7. Repetição de padrões na encriptação de blocos

Na encriptação simétrica de blocos, padrões presentes no texto claro podem gerar padrões identificáveis no criptograma. Este comportamento pode ser exemplificado pela encriptação simétrica no modo de operação ECB, no comando a seguir, onde a arte em ASCII de um disco flexível (adaptada de `https://www.asciiart.eu`) é encriptada com AES-128-ECB. A Figura 5.15 mostra, no criptograma, os padrões equivalentes aos encontrados no texto claro.

```
C:\>openssl enc -e -AES-128-ECB -in floppy.txt -out floppypattern.txt

C:\>type floppy.txt
```

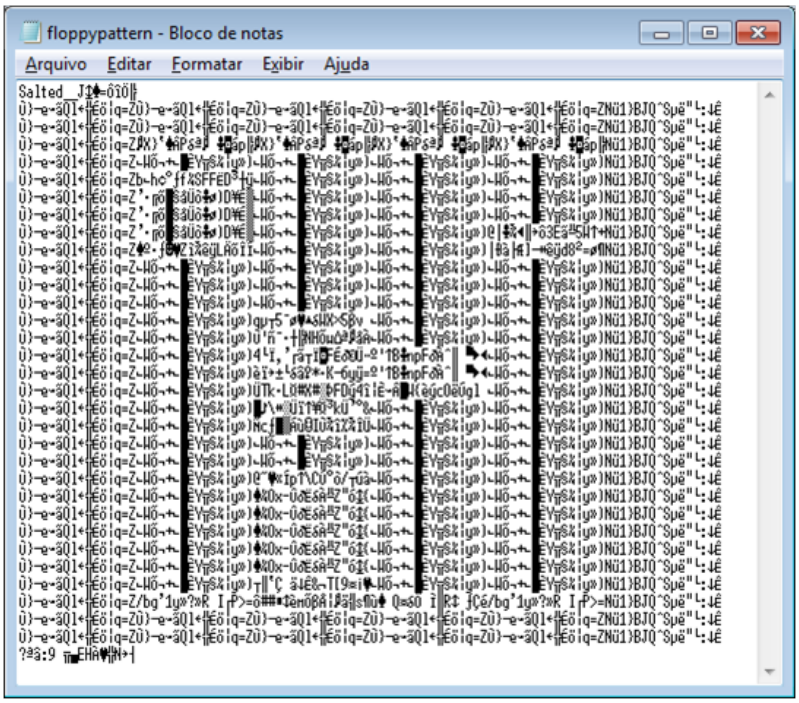


Figura 5.15. Padrões do texto claro geram padrões no criptograma com modo ECB.

5.4.8. Os modos de operação da encriptação simétrica de blocos

A Listagem 5.8 ilustra as diferenças de tolerância a falhas entre os modos de operação da encriptação simétrica, em particular, o algoritmo AES e os modos de operação ECB, CBC, CFB, OFB e CTR. O comando `type` (na linha 1) mostra o conteúdo do arquivo usado no teste. Todos os comandos de encriptação `enc -e` (linhas de 7 até 17) usam a mesma chave criptográfica, o mesmo arquivo de entrada e, quando necessário, o mesmo vetor de inicialização. A diferença entre eles está no modo de operação do AES: o comando da linha 4 usa o modo ECB (`-aes-128-ecb`), o comando da linha 7 usa o modo CBC (`-aes-128-cbc`), o comando da linha 10 usa o CFB (`-aes-128-cfb`), o comando da linha 13 usa o OFB (`-aes-128-ofb`) e a linha 10 usa o CTR (`-aes-128-ctr`).

Na linha 19, os arquivos de saída dos comandos `enc -e` são editados (`edit`), simulando a adulteração do criptograma: trocar o primeiro caractere pela próxima letra (p.ex., 'y' é substituído por 'z' e 'L' é substituído por 'M'), causando uma mudança no criptograma. Os comandos de decriptação `end -d` reagem de formas diferentes a adulteração: o comando no modo ECB (linha 21) perde todo o primeiro bloco do criptograma; o comando no modo CBC (linha 24) perde o primeiro bloco e o primeiro caractere do segundo bloco; o modo CFB (linha 28) perde o primeiro caractere do primeiro bloco e o segundo bloco inteiro; o modo OFB (linha 32) e o modo CTR (linha 36) perdem só o primeiro caractere do primeiro bloco.

Listagem 5.8. Os modos de operação e como toleram erros no criptograma.

```
1 C:\>type alex.txt
2 ALEXANDRE MELO BRAGA ALEXANDRE MELO BRAGA
3
4 C:\> openssl enc -e -a -aes-128-ecb -in alex.txt -K 6CB40CAEF6FDEDB31DBD908256F63276
5  -out alex.ecb
6
7 C:\> openssl enc -e -a -aes-128-cbc -in alex.txt -K 6CB40CAEF6FDEDB31DBD908256F63276
8  -iv E570548E2B1376147E3342F08082E29A -out alex.cbc
9
10 C:\> openssl enc -e -a -aes-128-cfb -in alex.txt -K 6CB40CAEF6FDEDB31DBD908256F63276
11  -iv E570548E2B1376147E3342F08082E29A -out alex.cfb
12
13 C:\> openssl enc -e -a -aes-128-ofb -in alex.txt -K 6CB40CAEF6FDEDB31DBD908256F63276
14 -iv E570548E2B1376147E3342F08082E29A -out alex.ofb
15
16 C:\> openssl enc -e -a -aes-128-ctr -in alex.txt -K 6CB40CAEF6FDEDB31DBD908256F63276
17  -iv E570548E2B1376147E3342F08082E29A -out alex.ctr
18
19 C:\>edit alex.ecb alex.cbc alex.cfb alex.ofb alex.ctr
20
21 C:\> openssl enc -d -a -aes-128-ecb -in alex.ecb -K 6CB40CAEF6FDEDB31DBD908256F63276
22 ****************RAGA ALEXANDRE MELO BRAGA
23
24 C:\> openssl enc -d -a -aes-128-cbc -in alex.cbc -K 6CB40CAEF6FDEDB31DBD908256F63276
25  -iv E570548E2B1376147E3342F08082E29A
26 ******************AGA ALEXANDRE MELO BRAGA
27
28 C:\> openssl enc -d -a -aes-128-cfb -in alex.cfb -K 6CB40CAEF6FDEDB31DBD908256F63276
29  -iv E570548E2B1376147E3342F08082E29A
30 ELEXANDRE MELO B****************ELO BRAGA
31
32 C:\> openssl enc -d -a -aes-128-ofb -in alex.ofb -K 6CB40CAEF6FDEDB31DBD908256F63276
33 -iv E570548E2B1376147E3342F08082E29A
34 *LEXANDRE MELO BRAGA ALEXANDRE MELO BRAGA
35
36 C:\> openssl enc -d -a -aes-128-ctr -in alex.ctr -K 6CB40CAEF6FDEDB31DBD908256F63276
37  -iv E570548E2B1376147E3342F08082E29A
38 *LEXANDRE MELO BRAGA ALEXANDRE MELO BRAGA
```

5.4.9. Funções do `OpenSSL` para cálculo de *hash* e MAC

A Listagem 5.9 mostra algumas maneiras de calcular valores *hash* e *tags* de autenticação MAC com o comando `dgst` do `OpenSSL`. Os primeiros quatro comandos calculam *hashes*. O comando `dgst` na linha 1 calcula o valor *hash* do arquivo de exemplo `gabarito.txt` com a função de resumo criptográfico SHA-256 (opção `-sha256`) e imprime no terminal o valor em hexadecimal (opção `-r`).

O comando `dgst` na linha 4 calcula o mesmo valor de *hash* do arquivo `gabarito.txt` com opção `-sha256`, mas não usa a opção `-r` e mostra o mesmo hexadecimal em uma linha formatada de modo diferente. Este exemplo ilustra o determinismo das funções de *hash* criptográfico. Isto é, mesmo texto claro de entrada e mesma função de *hash* resultam no mesmo valor *hash*, apesar das diferenças de apresentação.

O comando `dgst` na linha 7 calcula o valor de *hash* do mesmo arquivo `gabarito.txt` com a função de resumo criptográfico SHA-512 (opção `-sha512`), resultando em um valor diferente do anterior e com o dobro de tamanho.

O comando `dgst` na linha 11 calcula o valor de *hash* do mesmo arquivo `gabarito.txt` com outra função de resumo criptográfico de 512 bits, o SHA3 (opção `-sha3-512`), resultando em uma valor diferente do anterior, mas com o mesmo tamanho.

Os dois últimos comandos da Listagem 5.9 calculam *tags* de autenticação do tipo HMAC. O comando `dgst` da linha 15 calcula a *tag* de 256 bits para o arquivo `gabarito.txt` com as opções `-hmac` e `-sha256`. Vale observar que a opção `-hmac` é acompanhada pelo valor hexadecimal da chave criptográfica utilizada pela função HMAC. Esta chave pode ter sido gerada, por exemplo, pelo comando `rand` mostrado anteriormente.

Finalmente, o comando `dgst` da linha 19 calcula uma *tag* de 512 bits para o arquivo `gabarito.txt` com as opções `-sha512` e `-hmac`, utilizando a mesma chave criptográfica do exemplo anterior, e resulta em uma *tag* diferente da anterior, uma vez que a função de resumo criptográfico é diferente.

Listagem 5.9. Cálculo de *Hashes* e MACs.

```
1 C:\>openssl dgst -sha256 -r gabarito.txt
2 385fb55c7aed9ec76dc31b8392a90a61ba5acd1e578112a1977047f9d1e9c3b7 *gabarito.txt
3
4 C:\>openssl dgst -sha256 gabarito.txt
5 SHA256(gabarito.txt)= 385fb55c7aed9ec76dc31b8392a90a61ba5acd1e578112a1977047f9d1e9c3b7
6
7 C:\>openssl dgst -sha512 -r gabarito.txt
8 0cd7521e631ec426389e5ced1c9079486e5f733a6ab2545ff360333a259c9d3d9bb8659cfa32619df0e9dc
9 778d8a57988a681be5dc0d7942938fdefc9367b4ea *gabarito.txt
10
11 C:\>openssl dgst -sha3-512 -r gabarito.txt
12 2e3d039e704686479e7ad3145376232O4b68e1cb89739baa41a43650ac12f32b0bb76533a8045c5dde8a93
13 acb6743f8443b2c4dc43cdab10e78ab596167337a3 *gabarito.txt
14
15 C:\>openssl dgst -sha256 -hmac 3f44a88d098cdb8a384922e88a30dbe67f7178fd gabarito.txt
16 HMAC-SHA256(gabarito.txt)= f59848834988db68167a7a5527273ed494dbd6374cafefd0c5c3f7f18
17 dd7a02c
18
19 C:\>openssl dgst -sha3-512 -hmac 3f44a88d098cdb8a384922e88a30dbe67f7178fd gabarito.txt
20 HMAC-SHA3-512(gabarito.txt)= baf213dcc21d1bf9337be63495ceff4dd553c7cbb47da89818f7b9cf
21 ccdf336fe25c28c56e9f07062e74c83b57542b0df96223ed2af0765c9a4b678200160d80
```

5.4.10. Encriptação e decriptação assimétricas com RSA

Esta seção mostra a geração de um par de chaves assimétricas para o algoritmo RSA e dois exemplos de encriptação utilizado o mesmo par de chaves. Na Listagem 5.10, o comando `genrsa` na linha 1 gera chaves RSA de 2048 bits e usa a opção `-rand` para alimentar o gerador de números pseudo-aleatórios com uma semente calculada anteriormente. A opção `-rand`, quando bem utilizada, ajuda a evitar a geração de chaves RSA com parâmetros de baixa entropia, uma fonte comum de vulnerabilidades de implantação. Além disso, o par de chaves é gravado em um arquivo de saída (opção `-out chaves_rsa`) encriptado com AES (opção `-aes-256-ctr`) e chave derivada a partir de uma senha.

O comando `rsa` da linha 9 é usado para exportar a chave pública (opção `-pubout`) contida no par de chaves gerado anteriormente (opção `-in chaves_rsa`) em um arquivo separado (opção `-out chave_pub.rsa`). Já o comando `rsa` da linha 13 é usado para exportar a chave privada contida no par de chaves gerado anteriormente (opção `-in chaves_rsa`) em um arquivo separado (opção `-out chave_priv.rsa`). Vale observar a usabilidade ruim do `OpenSSL`: se a opção `-pubout` não é usada, o comportamento padrão (inseguro) é exportar a chave privada.

O comando `rsautl` da linha 17 encripta com RSA-OAEP (opções `-encrypt -oaep`) e a chave pública (opção `-inkey chave_pub.rsa`). O comando `rsautl` da linha 20 decripta (`-decrypt -oaep`) com a chave privada (`-inkey chave_priv.rsa`).

Na linha 23, o comando `req` extrai a chave pública de um certificado auto-assinado. Esta chave é usada no comando `smime` da linha 20 para encriptar uma chave simétrica temporária (opção `-aes-128-cbc`) que encripta o texto claro. Já a linha 27 decripta com a chave privada a chave temporária embutida no criptograma que é usada para decriptar o texto claro.

Listagem 5.10. Encriptação com RSA-OAEP e chave de transporte SMIME.

```
1 C:\>openssl genrsa -rand random.txt -aes-256-ctr -out chaves_rsa 2048
2 Generating RSA private key, 2048 bit long modulus (2 primes)
3 ..................................................+++++
4 .............+++++
5 e is 65537 (0x010001)
6 Enter pass phrase for chaves_rsa:
7 Verifying - Enter pass phrase for chaves_rsa:
8
9 C:\>openssl rsa -in chaves_rsa -pubout -out chave_pub.rsa
10 Enter pass phrase for chaves_rsa:
11 writing RSA key
12
13 C:\>openssl rsa -in chaves_rsa -out chave_priv.rsa
14 Enter pass phrase for chaves_rsa:
15 writing RSA key
16
17 C:\>openssl rsautl -encrypt -oaep -in gabarito.txt -pubin -inkey chave_pub.rsa
18 -out gabarito.rsac
19
20 C:\>openssl rsautl -decrypt -oaep -in gabarito.rsac -inkey chave_priv.rsa
21  -out gabarito6.txt
22
23 C:\>openssl req -new -key chaves_rsa -x509 -out raiz.crt
24
25 C:\>openssl smime -in gabarito.txt -encrypt -out gabarito.smime -aes-128-cbc raiz.crt
26
27 C:\>openssl smime -decrypt -in gabarito.smime -inkey chave_priv.rsa
28 Gabarito da prova
29 1-a 2-c 3-d 4-c 5-a 6-b 7-d 8-d 9-a 10-c
```

5.4.11. Duas versões de RSA PKCS#1: v1.5 e v2.0

A Listagem 5.11 mostra dois exemplos de encriptação assimétrica com RSA. O primeiro sem a opção `-oaep` e o segundo com este opção explícita. A ausência da opção `-oaep` na encriptação RSA com o comando `rsautl` faz com que o preenchimento (*padding*) PKCS#1 v1.5 seja utilizado. Este formato de preenchimento é, em certos casos, considerado inseguro [Bleichenbacher 1998], devido a sua susceptibilidade ao ataque de *padding oracle*, mas que ainda é bastante utilizado no SSL/TLS até a versão v1.2, Mostrando como é difícil seguir as boas práticas e ao mesmo tempo manter a compatibilidade com o passado.

Tanto a PKCS#1 v1.5, quanto o RSA-OAEP (PKCS#1 v2.0) limitam o tamanho do texto claro que pode ser encriptado em uma única chamada do comando `rsautl`. Este limite está relacionado ao tamanho do corpo finito (e da chave) usado na aritmética modular do algoritmo RSA e, no cado o RSA-OAEP, pode ser determinado, em bytes, pela fórmula *(ks-2*hs)/8–2*, onde *ks* é o tamanho da chave RSA em bits e *hs* é o tamanho do *hash* em bits usado pelo *padding* OAEP.

Na Listagem 5.11, os comandos das linhas 5 e 6 terminam com sucesso a encriptação do arquivo curto, enquanto os comandos de encriptação do arquivo médio falham em parte, por que o comando da linha 16 (com `-oaep`) já não consegue processar tantos dados de entrada. Já ambos os comandos de encriptação do arquivo longo falham devido ao excesso de dados de entrada (apenas 50 caracteres a mais que no caso anterior), conforme as mensagens de erro emitidas.

Listagem 5.11. Limite de tamanho para o criptograma no RSA.

```
1 C:\>type curto.txt
2 Gabarito da prova:
3 01-a 02-c 03-d 04-c 05-a 06-b 07-d 08-d 09-a 10-c
4
5 C:\>openssl rsautl -encrypt -in curto.txt -pubin -inkey chave_pub.rsa -out o.rsac
6 C:\>openssl rsautl -encrypt -oaep -in curto.txt -pubin -inkey chave_pub.rsa -out o.oaep
7
8 C:\>type medio.txt
9 Gabarito da prova:
10 01-a 02-c 03-d 04-c 05-a 06-b 07-d 08-d 09-a 10-c
11 11-a 12-c 13-d 14-c 15-a 16-b 17-d 18-d 19-a 20-c
12 21-a 22-c 23-d 24-c 25-a 26-b 27-d 28-d 29-a 30-c
13 31-a 32-c 33-d 34-c 35-a 36-b 37-d 38-d 39-a 40-c
14
15 C:\>openssl rsautl -encrypt -in medio.txt -pubin -inkey chave_pub.rsa -out o.rsac
16 C:\>openssl rsautl -encrypt -oaep -in medio.txt -pubin -inkey chave_pub.rsa -out o.oaep
17 RSA operation error
18 2636:error:0409A06E:rsa routines:RSA_padding_add_PKCS1_OAEP_mgf1
19 :data too large for key size:crypto\rsa\rsa_oaep.c:62:
20
21 C:\>type longo.txt
22 Gabarito da prova:
23 01-a 02-c 03-d 04-c 05-a 06-b 07-d 08-d 09-a 10-c
24 11-a 12-c 13-d 14-c 15-a 16-b 17-d 18-d 19-a 20-c
25 21-a 22-c 23-d 24-c 25-a 26-b 27-d 28-d 29-a 30-c
26 31-a 32-c 33-d 34-c 35-a 36-b 37-d 38-d 39-a 40-c
27 41-a 42-c 43-d 44-c 45-a 46-b 47-d 48-d 49-a 50-c
28 C:\>openssl rsautl -encrypt -in longo.txt -pubin -inkey chave_pub.rsa -out o.rsac
29 RSA operation error
30 6504:error:0406D06E:rsa routines:RSA_padding_add_PKCS1_type_2
31 :data too large for key size:crypto\rsa\rsa_pk1.c:125:
32
33 C:\>openssl rsautl -encrypt -oaep -in longo.txt -pubin -inkey chave_pub.rsa -out o.rsac
34 RSA operation error
35 8988:error:0409A06E:rsa routines:RSA_padding_add_PKCS1_OAEP_mgf1
36 :data too large for key size:crypto\rsa\rsa_oaep.c:62:
```

5.4.12. Geração de certificados digitais

A Listagem 5.12 contém os comandos do `OpenSSL` para gerar um par de chaves RSA, gerar um certificado digital auto-assinado para a chave pública do par (ilustrando uma Autoridade Certificadora - CA), gerar um par de chaves para um usuário final e assinar o certificado digital deste usuário com a chave privada da CA.

O comando `genrsa` da linha 1 gera as chaves de 2048 bits da CA e as armazena em um arquivo encriptado com `-aes-128-cbc` e uma chave derivada de senha. Já o comando `req -new` da linha 9 gera uma requisição para um certificado X.509 auto-assinado da chave pública da CA. O outro comando `genrsa` da linha 13 gera um par de chaves RSA de 2048 bits para um usuário final, enquanto o comando `req -new` da linha 21 gera uma requisição de certificado para a chave pública do usuário. Uma vez que a validade não foi informada, estes certificados tem validade *default* de um mês (30 dias).

Finalmente, o comando `x509` da linha 25 recebe a requisição de assinatura de certificado (opção `-in alex.csr`), e a assina com a chave privada da CA (opção `-CAkey CA_chaves_rsa`), resultado no certificado digital do usuário final (opção `-out alex.crt`) com número de série 3. Os comandos `verify` das linhas 32 e 35 verificam as assinaturas digitais, respectivamente, dos certificados da CA e do usuário final.

Listagem 5.12. Certificados digitais para chaves RSA.

```
1 C:\>openssl genrsa -aes-128-cbc -out CA_chaves_rsa
2 Generating RSA private key, 2048 bit long modulus (2 primes)
3 ......+++++
4 ..............................................................+++++
5 e is 65537 (0x010001)
6 Enter pass phrase for CA_chaves_rsa:
7 Verifying - Enter pass phrase for CA_chaves_rsa:
8
9 C:\>openssl req -new -key CA_chaves_rsa -x509 -out CAraiz.crt
10 Enter pass phrase for CA_chaves_rsa:
11 #[]...
12
13 C:\>openssl genrsa -aes-128-cbc -out alex_rsa
14 Generating RSA private key, 2048 bit long modulus (2 primes)
15 ......+++++
16 ..............+++++
17 e is 65537 (0x010001)
18 Enter pass phrase for alex_rsa:
19 Verifying - Enter pass phrase for alex_rsa:
20
21 C:\>openssl req -new -key alex_rsa -out alex.csr
22 Enter pass phrase for alex_rsa:
23 #...
24
25 C:\>openssl x509 -req -in alex.csr -CA CAraiz.crt -CAkey CA_cha
26 ves_rsa -out alex.crt -set_serial 3
27 Signature ok
28 subject=C = br, ST = sp, L = cps, O = alex, CN = www.alex.com.br
29 Getting CA Private Key
30 Enter pass phrase for CA_chaves_rsa:
31
32 C:\>openssl verify -CAfile CAraiz.crt CAraiz.crt
33 CAraiz.crt: OK
34
35 C:\>openssl verify -CAfile CAraiz.crt alex.crt
36 alex.crt: OK
```

5.4.13. Assinaturas digitais com RSA

A Listagem 5.13 mostra os comandos do `OpenSSL` para geração de assinaturas digitais e verificação destas assinaturas. Neste exemplo, a assinatura é computada com o algoritmo RSA e chaves de 2048 bits sobre o *hash* do arquivo de entrada.

O comando `dgst` na linha 1 gera uma assinatura digital do arquivo `gabarito.txt` com a chave privada (opção `-sign alex_rsa`). A assinatura é gerada sobre um *hash* de 512 bits do arquivo de entrada (opção `-sha512`). O resultado é mostrado no terminal em hexadecimal (opção `-hex`), onde se vê que a assinatura tem 2048 bits de tamanho.

O comando `dgst` da linha 11 repete o processo de assinatura do comando anterior, mostrando que este método de assinatura digital com RSA é determinístico. Isto é, se forem fornecidos o mesmo arquivo de entrada, a mesma chave privada e a mesma função de *hash*, então a mesma assinatura será gerada.

Já o comando `dgst` da linha 21 não usa a opção `-hex` e grava a assinatura em arquivo de saída. O comando `rsa` da linha 24 exporta a chave pública em um arquivo separado para que ela seja usada na verificação de assinaturas geradas pela chave privada correspondente. Finalmente, na linha 26, o comando `dgst` usa a chave pública para verificar (opção `-verify alex_pub`) a assinatura (opção `-signature gabarito.sign`) do arquivo de entrada `gabarito.txt`. A verificação é bem-sucedida.

Listagem 5.13. Assinaturas digitais e verificação de assinaturas com RSA.

```
1 C:\>openssl dgst -sha512 -hex -sign alex_rsa gabarito.txt
2 Enter pass phrase for alex_rsa:
3 RSA-SHA512(gabarito.txt)= 0b84698f3e0ef7279638819731532e2f65f84e7013cab1193f0582
4 84f022433d90cc5978ea3cc8486662fb48884d072a0ec1e0cf4ddb75f8503ff3f564125303d4d861
5 b8bd718e5eab85b5902f1881845d11c417d1c5796959e5d4baee4da3216e0ba210d6ce583ca7ea53
6 83f9ab727991a7e6404babc7cfe2fc28885e126a36db28a502f2ae76dde734086686806550c5e20c
7 e2fe1bea67a09b6caf62c176df40a56d54470f86b739539d3e5409dc00a92324cedd4737228b653c
8 18ee84d2b81b49eabdb97ff2e7d802a44eaa3de12c6b684e4be88a81008e9bb776c4e3c1e9144e0c
9 c0acbc702ff35904d2b30c0e2e0a1e26344ce6cb2fd96c1e8f3ecbba11
10
11 C:\>openssl dgst -sha512 -hex -sign alex_rsa gabarito.txt
12 Enter pass phrase for alex_rsa:
13 RSA-SHA512(gabarito.txt)= 0b84698f3e0ef7279638819731532e2f65f84e7013cab1193f0582
14 84f022433d90cc5978ea3cc8486662fb48884d072a0ec1e0cf4ddb75f8503ff3f564125303d4d861
15 b8bd718e5eab85b5902f1881845d11c417d1c5796959e5d4baee4da3216e0ba210d6ce583ca7ea53
16 83f9ab727991a7e6404babc7cfe2fc28885e126a36db28a502f2ae76dde734086686806550c5e20c
17 e2fe1bea67a09b6caf62c176df40a56d54470f86b739539d3e5409dc00a92324cedd4737228b653c
18 18ee84d2b81b49eabdb97ff2e7d802a44eaa3de12c6b684e4be88a81008e9bb776c4e3c1e9144e0c
19 c0acbc702ff35904d2b30c0e2e0a1e26344ce6cb2fd96c1e8f3ecbba11
20
21 C:\>openssl dgst -sha512 -sign alex_rsa -out gabarito.sign  gabarito.txt
22 Enter pass phrase for alex_rsa:
23
24 C:\>openssl rsa -in alex_rsa -pubout -out alex_pub
25
26 C:\>openssl dgst -sha512 -verify alex_pub -signature gabarito.sign gabarito.txt
27 Verified OK
```

5.4.14. Curvas elípticas no `OpenSSL`

O `OpenSSL` oferece diversas opções de curvas elípticas. A Listagem 5.14 contém as curvas elípticas sobre corpos primos F_p e binários F_{2^m} disponíveis no `OpenSSL`, e inclui aquelas padronizadas e ainda consideradas seguras pelos padrões internacionais [NIST 2013, SEC 2010]. Algumas curvas foram omitidas por falta de espaço. A nomenclatura das curvas segue o padrão SEC-2 [SEC 2010], com nomes no formato `sec[p|t]XXX[r|k]v`, onde `[p|t]` indica uma curva sobre corpo primo (`p`) ou binário (`t`), enquanto o número `XXX` indica o tamanho em bits da chave e a letra `[r|k]` indica que a curva usa os parâmetros de Koblitz (`k`) ou randômicos (`r`), em uma determinada versão (`v`).

A versão do `OpenSSL` usada neste texto não contém a curva 25519 [Bernstein 2006] disponível na CLI. Além disto, na Listagem 5.14, as linhas marcadas com sinal de mais (+) indicam as curvas presentes no padrão SEC-2 [SEC 2010]. Já as linhas marcadas com sinal de menos (-) indicam as curvas que são consideradas fracas para os padrões de segurança atuais, com menos de 80 bits de segurança. Por exemplo, estas curvas fizeram parte da versão 1 do padrão SEC-2 [SEC 2000], mas que foram excluídas da versão 2 deste mesmo padrão [SEC 2010]. As curvas marcadas com um asterisco (*) são consideradas inseguras por Daniel Bernstein [Bernstein et al. 2013], que considera estas curvas complexas de implementar, facilitando a ocorrência de defeitos que levam a vulnerabilidades, entre outros problemas.

As curvas fracas ou inseguras não devem mais ser usadas. As curvas com nível de segurança menor ou igual a 80 bits [SEC 2000] são as seguintes: secp112r1, secp112r2, secp128r1, secp128r2, secp160k1, secp160r1, secp160r2, sect113r1, sect113r2 ,sect131r1 e sect131r2. As curvas inseguras [Bernstein et al. 2013] são as seguintes: secp224r1, secp256k1 ,secp384r1, brainpoolP256t1 e brainpoolP384t1. Excluindo-se estas exceções, em princípio, as outras curvas elípticas na Listagem 5.14 podem ser utilizadas.

Listagem 5.14. As curvas elípticas disponíveis no `OpenSSL`.

```
1 C:\>openssl ecparam -list_curves
2  -  secp112r1 : SECG/WTLS curve over a 112 bit prime field
3  -  secp112r2 : SECG curve over a 112 bit prime field
4  -  secp128r1 : SECG curve over a 128 bit prime field
5  -  secp128r2 : SECG curve over a 128 bit prime field
6  -  secp160k1 : SECG curve over a 160 bit prime field
7  -  secp160r1 : SECG curve over a 160 bit prime field
8  -  secp160r2 : SECG/WTLS curve over a 160 bit prime field
9  +  secp192k1 : SECG curve over a 192 bit prime field
10 +  secp224k1 : SECG curve over a 224 bit prime field
11 *+ secp224r1 : NIST/SECG curve over a 224 bit prime field
12 *+ secp256k1 : SECG curve over a 256 bit prime field
13 *+ secp384r1 : NIST/SECG curve over a 384 bit prime field
14 +  secp521r1 : NIST/SECG curve over a 521 bit prime field
15    prime192v1: NIST/X9.62/SECG curve over a 192 bit prime field
16    prime192v2: X9.62 curve over a 192 bit prime field
17    prime192v3: X9.62 curve over a 192 bit prime field
18    prime239v1: X9.62 curve over a 239 bit prime field
19    prime239v2: X9.62 curve over a 239 bit prime field
20    prime239v3: X9.62 curve over a 239 bit prime field
21    prime256v1: X9.62/SECG curve over a 256 bit prime field
22 -  sect113r1 : SECG curve over a 113 bit binary field
23 -  sect113r2 : SECG curve over a 113 bit binary field
24 -  sect131r1 : SECG/WTLS curve over a 131 bit binary field
25 -  sect131r2 : SECG curve over a 131 bit binary field
26 +  sect163k1 : NIST/SECG/WTLS curve over a 163 bit binary field
27 +  sect163r1 : SECG curve over a 163 bit binary field
28 +  sect163r2 : NIST/SECG curve over a 163 bit binary field
29    sect193r1 : SECG curve over a 193 bit binary field
```

```
30      sect193r2 : SECG curve over a 193 bit binary field
31  +   sect233k1 : NIST/SECG/WTLS curve over a 233 bit binary field
32  +   sect233r1 : NIST/SECG/WTLS curve over a 233 bit binary field
33  +   sect239k1 : SECG curve over a 239 bit binary field
34  +   sect283k1 : NIST/SECG curve over a 283 bit binary field
35  +   sect283r1 : NIST/SECG curve over a 283 bit binary field
36  +   sect409k1 : NIST/SECG curve over a 409 bit binary field
37  +   sect409r1 : NIST/SECG curve over a 409 bit binary field
38  +   sect571k1 : NIST/SECG curve over a 571 bit binary field
39  +   sect571r1 : NIST/SECG curve over a 571 bit binary field
40      c2pnb163v1: X9.62 curve over a 163 bit binary field
41      c2pnb163v2: X9.62 curve over a 163 bit binary field
42      c2pnb163v3: X9.62 curve over a 163 bit binary field
43      c2pnb176v1: X9.62 curve over a 176 bit binary field
44      c2tnb191v1: X9.62 curve over a 191 bit binary field
45      c2tnb191v2: X9.62 curve over a 191 bit binary field
46      c2tnb191v3: X9.62 curve over a 191 bit binary field
47      c2pnb208w1: X9.62 curve over a 208 bit binary field
48      c2tnb239v1: X9.62 curve over a 239 bit binary field
49      c2tnb239v2: X9.62 curve over a 239 bit binary field
50      c2tnb239v3: X9.62 curve over a 239 bit binary field
51      c2pnb272w1: X9.62 curve over a 272 bit binary field
52      c2pnb304w1: X9.62 curve over a 304 bit binary field
53      c2tnb359v1: X9.62 curve over a 359 bit binary field
54      c2pnb368w1: X9.62 curve over a 368 bit binary field
55      c2tnb431r1: X9.62 curve over a 431 bit binary field
56      [ ... ]
57      brainpoolP224t1: RFC 5639 curve over a 224 bit prime field
58      brainpoolP256r1: RFC 5639 curve over a 256 bit prime field
59  *   brainpoolP256t1: RFC 5639 curve over a 256 bit prime field
60      brainpoolP320r1: RFC 5639 curve over a 320 bit prime field
61      brainpoolP320t1: RFC 5639 curve over a 320 bit prime field
62      brainpoolP384r1: RFC 5639 curve over a 384 bit prime field
63  *   brainpoolP384t1: RFC 5639 curve over a 384 bit prime field
64      brainpoolP512r1: RFC 5639 curve over a 512 bit prime field
65      brainpoolP512t1: RFC 5639 curve over a 512 bit prime field
66      SM2         : SM2 curve over a 256 bit prime field
```

A criptografia de curvas elípticas ainda é uma área de pesquisa bastante ativa na qual existe uma grande variedade de curvas elípticas propostas para usos criptográficos, ao mesmo tempo em que há também esforços para comprovar ou refutar as alegações de segurança feitas para determinadas curvas. Por isto, é importante observar as fontes supracitadas em busca de atualizações de segurança.

5.4.15. Assinaturas digitais com ECDSA

Esta seção mostra os comandos `OpenSSL` usados para selecionar curvas elípticas, gerar pares de chaves criptográficas a partir das curvas selecionadas, gerar assinaturas digitais com o algoritmo ECDSA e verificar estas assinaturas.

Na Listagem 5.2, os comandos `ecparam -name`, nas linhas 1 e 2, criam parâmetros criptográficos para as duas curvas elípticas secp521r1 e prime256v1, respectivamente, e gravam-nos em arquivos de saída. Já os dois comandos `ecparam -genkey` nas linhas 4 e 5 geram os pares de chaves criptográficas para as duas curvas selecionadas anteriormente. Nas linhas 7 e 11, as chaves públicas dos dois pares de chaves são exportadas em arquivos separados pelos comandos `ec -pubout`.

Além disso, nas linhas 15 e 16, o comando `dgst` é usado para gerar duas assinaturas digitais sobre o *hash* (opção `-sha512`) do mesmo arquivo de entrada `gabarito.txt`, resultando em uma assinatura para cada curva selecionada neste exemplo. O comando `dgst`

`-sha512 -verify` da linha 18 verifica a assinatura digital gerada com a curva secp512r1, enquanto o comando `dgst -sha512 -verify` análogo, na linha 21, verifica a assinatura digital gerada com a curva prime256v1. As duas verificações são bem-sucedidas.

Finalmente, os comandos `ecparam` das linhas 24 e 28 mostram duas maneiras diferentes de descobrir informações de uma curva a partir dos parâmetros. A primeira lista os nomes da curva. A segunda, mais detalhada, lista em detalhe os parâmetros da curva em hexadecimal: primo, *a*, *b*, gerador, ordem, cofator e semente.

Listagem 5.15. Assinaturas digitais e verificação de assinaturas com ECDSA.

```
1 C:\>openssl ecparam -name  secp521r1 -out ecparam1.pem
2 C:\>openssl ecparam -name prime256v1 -out ecparam2.pem
3
4 C:\>openssl ecparam -genkey -in ecparam1.pem -noout -out eckeys1.pem
5 C:\>openssl ecparam -genkey -in ecparam2.pem -noout -out eckeys2.pem
6
7 C:\>openssl ec -in eckeys1.pem -pubout -out public1.pem
8 read EC key
9 writing EC key
10
11 C:\>openssl ec -in eckeys2.pem -pubout -out public2.pem
12 read EC key
13 writing EC key
14
15 C:\>openssl dgst -SHA512 -sign eckeys1.pem -out signature1.sign gabarito.txt
16 C:\>openssl dgst -SHA512 -sign eckeys2.pem -out signature2.sign gabarito.txt
17
18 C:\>openssl dgst -SHA512 -verify public1.pem -signature signature1.sign gabarito.txt
19 Verified OK
20
21 C:\>openssl dgst -SHA512 -verify public2.pem -signature signature2.sign gabarito.txt
22 Verified OK
23
24 C:\>openssl ecparam -in ecparam2.pem -text -noout
25 ASN1 OID: prime256v1
26 NIST CURVE: P-256
27
28 C:\>openssl ecparam -in ecparam2.pem -text -param_enc explicit -noout
29 Field Type: prime-field
30 Prime:
31     00:ff:ff:ff:ff:00:00:00:01:00:00:00:00:00:00:
32     00:00:00:00:00:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:
33     ff:ff:ff
34 A:
35     00:ff:ff:ff:ff:00:00:00:01:00:00:00:00:00:00:
36     00:00:00:00:00:00:ff:ff:ff:ff:ff:ff:ff:ff:ff:
37     ff:ff:fc
38 B:
39     5a:c6:35:d8:aa:3a:93:e7:b3:eb:bd:55:76:98:86:
40     bc:65:1d:06:b0:cc:53:b0:f6:3b:ce:3c:3e:27:d2:
41     60:4b
42 Generator (uncompressed):
43     04:6b:17:d1:f2:e1:2c:42:47:f8:bc:e6:e5:63:a4:
44     40:f2:77:03:7d:81:2d:eb:33:a0:f4:a1:39:45:d8:
45     98:c2:96:4f:e3:42:e2:fe:1a:7f:9b:8e:e7:eb:4a:
46     7c:0f:9e:16:2b:ce:33:57:6b:31:5e:ce:cb:b6:40:
47     68:37:bf:51:f5
48 Order:
49     00:ff:ff:ff:ff:00:00:00:00:ff:ff:ff:ff:ff:ff:
50     ff:ff:bc:e6:fa:ad:a7:17:9e:84:f3:b9:ca:c2:fc:
51     63:25:51
52 Cofactor:  1 (0x1)
53 Seed:
54     c4:9d:36:08:86:e7:04:93:6a:66:78:e1:13:9d:26:
55     b7:81:9f:7e:90
```

5.5. Avaliação de segurança criptográfica do TLS

Esta seção é organizada em torno de avaliações de segurança do SSL/TLS para detecção de suas vulnerabilidades nos servidores web por meio de comandos do `OpenSSL`, assim como também por outras ferramentas de análise de segurança, tais como, por exemplo: o `sslscan` [rbsec 2019], o `TestSSLServer` [Pornin 2017], o `SSLlabs` [Qualys 2019] e o `Observatory` [mozilla 2019], entre outras. Nesta seção, o `OpenSSL` é usado como uma ferramenta de testes e não como uma infraestrutura criptográfica, contrastando com a seção anterior.

5.5.1. Testando uma conexão TLS com `OpenSSL`

O `OpenSSL` vem com uma ferramenta cliente TLS que pode ser usada para estabelecer conexões com um servidor. O primeiro comando na Listagem 5.16 mostra como o cliente `s_client` usa a opção `-connect`, o nome do servidor e a porta 443 (padrão do SSL) para estabelecer uma conexão segura (na versão 1.2 `-tls1.2`) com um servidor web.

O suporte aos algoritmos criptográficos específicos pode ser testado pela opção `-cipher` do comando `s_client` seguida do nome da *suite* criptográfica. Já o comando `ciphers -s` lista os algoritmos suportados na instalação local no `OpenSSL`. Esta lista é ligeiramente diferente daquela mostrada por outras opções do comando `ciphers` analisadas adiante no texto.

Os outros comandos na Listagem 5.16 mostram os algoritmos disponíveis e depois testam a conexão com criptografias específicas: `-cipher AES128-SHA256` e `-cipher kECDHE`, que valida o uso do ECDH efêmero no acordo de chaves.

Se o endereço `exemplo.com` for substituído por uma URL verdadeira, o comando `s_client` emite informações diagnósticas da conexão com o servidor. O `s_client` espera que um comando HTTP seja fornecido para o servidor web. Uma maneira simples de fazer isto é digitar no *prompt* do terminal o comando HTTP `HEAD / HTTP/1.0` seguido opcionalmente da URL do servidor.

As informações de diagnóstico incluem dados da cadeia de certificação, o certificado do servidor, a versão do protocolo, a *suite* criptográfica, o tamanho da chave pública do servidor, o estado das flags de renegociação e de compressão e várias informações da sessão SSL.

Listagem 5.16. O `OpenSSL` como um cliente TLS.

```
1 C:\>openssl s_client -connect exemplo.com:443 -tls1_2
2
3 C:\>openssl ciphers -s
4 TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:
5 ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:
6 ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-CHACHA20-POLY1305:
7 ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256:
8 ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES256-SHA256:
9 ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA256:
10 ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA:
11 ECDHE-RSA-AES128-SHA:DHE-RSA-AES128-SHA:AES256-GCM-SHA384:AES128-GCM-SHA256:
12 AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-SHA
13
14 C:\>openssl s_client -connect exemplo.com -port 443 -cipher AES128-SHA256
15
16 C:\>openssl s_client -connect exemplo.com -port 443 -cipher kECDHE
```

5.5.2. Testando o suporte do servidor às versões do SSL/TLS

O comportamento padrão do comando `s_client` é tentar estabelecer uma conexão na versão mais alta do protocolo disponível na instalação do `OpenSSL`. Versões específicas podem ser usadas explicitamente com as opções `-ssl2` (disponível apenas em versões antigas do `OpenSSL`), `-ssl3`, `-tls1`, `-tls1_1`, `-tls1_2` e `-tls1_3` (somente em distribuições novas). Além disso, versões do TLS podem ser excluídas explicitamente com as opções `-no_ssl2`, `-no_ssl3`, `-no_tls1`, `-no_tls1_1`, `-no_tls1_2`.

A Listagem 5.17 mostra o resultado da tentativa mal-sucedida do cliente `s_client -connect` em estabelecer uma conexão com um servidor web na versão 1.3 do TLS (opção `-tls1_3`). Este falha de conexão tende a se tornar menos comum à medida que mais servidores adotem a nova versão.

Listagem 5.17. Exemplo de servidor web sem suporte ao TLS 1.3.

```
C:\>openssl s_client -connect www.exemplo.com -port 443 -tls1_3
CONNECTED(000000E8)
5140:error:14094410:SSL routines:ssl3_read_bytes:sslv3 alert handshake failure:s
sl\record\rec_layer_s3.c:1528:SSL alert number 40
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 237 bytes
Verification: OK
---
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
```

O comando `ciphers -v "ALL"` (sem filtro) mostra em mais detalhe uma lista longa das *suites* criptográficas do SSL/TLS, que pode ser filtrada para versões ou algoritmos específicos. A Listagem 5.18 mostra o resultado do comando `ciphers -v "ALL"` filtrado apenas para as *suites* do TLSv1 que contém o acordo de chaves ECDH.

Os itens desta lista podem ser testados contra um servidor web com o comando `s_client -connect`, conforme exemplificado ao final da Listagem 5.18.

Listagem 5.18. Testes por *suites* criptográficas específicas.

```
C:\>openssl ciphers -v "ALL" | find "TLSv1 " | find "ECDH"
ECDHE-ECDSA-AES256-SHA   TLSv1 Kx=ECDH     Au=ECDSA Enc=AES(256)  Mac=SHA1
ECDHE-RSA-AES256-SHA     TLSv1 Kx=ECDH     Au=RSA   Enc=AES(256)  Mac=SHA1
AECDH-AES256-SHA         TLSv1 Kx=ECDH     Au=None  Enc=AES(256)  Mac=SHA1
ECDHE-ECDSA-AES128-SHA   TLSv1 Kx=ECDH     Au=ECDSA Enc=AES(128)  Mac=SHA1
ECDHE-RSA-AES128-SHA     TLSv1 Kx=ECDH     Au=RSA   Enc=AES(128)  Mac=SHA1
AECDH-AES128-SHA         TLSv1 Kx=ECDH     Au=None  Enc=AES(128)  Mac=SHA1
ECDHE-PSK-AES256-CBC-SHA384 TLSv1 Kx=ECDHEPSK Au=PSK  Enc=AES(256)  Mac=SHA384
ECDHE-PSK-AES256-CBC-SHA TLSv1 Kx=ECDHEPSK Au=PSK   Enc=AES(256)  Mac=SHA1
ECDHE-PSK-CAMELLIA256-SHA384 TLSv1 Kx=ECDHEPSK Au=PSK  Enc=Camellia(256) Mac=SHA384
ECDHE-PSK-AES128-CBC-SHA256 TLSv1 Kx=ECDHEPSK Au=PSK  Enc=AES(128)  Mac=SHA256
ECDHE-PSK-AES128-CBC-SHA TLSv1 Kx=ECDHEPSK Au=PSK   Enc=AES(128)  Mac=SHA1
ECDHE-PSK-CAMELLIA128-SHA256 TLSv1 Kx=ECDHEPSK Au=PSK  Enc=Camellia(128) Mac=SHA256

C:\>openssl s_client -connect www.exemplo.com:443 -cipher ECDHE-ECDSA-AES256-SHA
```

5.5.3. Reuso de conexões e renegociação segura

A capacidade de reconectar utilizando parâmetros já calculados anteriormente é importante na garantia de disponibilidade de um servidor SSL/TLS. Por outro lado, as conexões SSL/TLS entre cliente e servidor web mantém parâmetros que devem ser recalculados periodicamente, caso contrário a sua utilização por tempo indeterminado ou por uma quantidade grande de reconexões traz insegurança tanto para o cliente quanto para o servidor. Assim, há um compromisso entre a quantidade de reconexões e a insegurança de reutilização de parâmetros.

A Listagem 5.19 mostra como a opção `-reconnect` do comando `s_client` pode ser usada para determinar quantas reconexões um servidor web permite para uma conexão SSL/TLS. Conforme ilustrado na Listagem 5.19, cinco reusos são esperados antes de uma nova conexão (omitida no texto juntamente com o restante da saída).

Listagem 5.19. Teste de reuso de conexões.

```
1 C:\>openssl s_client -connect www.exemplo.com:443 -reconnect | find /I "Reuse"
2 depth=2 C = GB, ST = Greater Manchester
      , L = Salford, O = COMODO CA Limited, CN = COMODO RSA Certification Authority
3 verify error:num=20:unable to get local issuer certificate
4 Reused, TLSv1.2, Cipher is ECDHE-RSA-AES128-GCM-SHA256
5 Reused, TLSv1.2, Cipher is ECDHE-RSA-AES128-GCM-SHA256
6 Reused, TLSv1.2, Cipher is ECDHE-RSA-AES128-GCM-SHA256
7 Reused, TLSv1.2, Cipher is ECDHE-RSA-AES128-GCM-SHA256
8 Reused, TLSv1.2, Cipher is ECDHE-RSA-AES128-GCM-SHA256
9 read:errno=0
```

A renegociação segura de parâmetros de conexão pode ser detectada facilmente nas versões novas do TLS. A Listagem 5.20 mostra o trecho da saída de um comando `s_client -connect` em que se pode observar a frase afirmativa *"Secure Renegotiation IS supported"* indicando o suporte a este característica. O contrário seria indicado explicitamente pela frase negativa.

Mesmo um servidor que suporta renegociação ainda pode rejeitar a renegociação iniciada pelo cliente, por que ela aumenta a superfície de ataque do servidor, abrindo oportunidade para ataques de *downgrade* sobre o protocolo. Dito isto, o `s_client -connect` pode solicitar a renegociação de parâmetros simplesmente digitando "R"na linha de comando do terminal (linhas 10 e 11). Se a renegociação for possível, o servidor reinicia o protocolo de *handshake* e envia novamente seu certificado. Se a renegociação iniciada pelo cliente não for possível, uma mensagem de erro (parecida com a mostrada nas linhas 13 e 14) é enviada pelo servidor.

Listagem 5.20. Renegociação segura.

```
1 C:\>openssl s_client -connect www.exemplo.com:443
2 [...]
3 New, TLSv1.2, Cipher is ECDHE-RSA-AES128-GCM-SHA256
4 Server public key is 2048 bit
5 Secure Renegotiation IS supported
6 Compression: NONE
7 Expansion: NONE
8 [...]
9
10 HEAD / HTTP/1.0
11 R
12 RENEGOTIATING
13 5620:error:1409444C:SSL routines:ssl3_read_bytes: tlsv1 alert no renegotiation:
14 ssl\record\rec_layer_s3.c:1528: SSL alert number 100
```

5.5.4. Verificação de certificados revogados em tempo real

A verificação de revogação de um certificado não precisa ser feita apenas por meio de CRLs distribuídas por ACs. Uma alternativa é um cliente SSL/TLS usar o *Online Certificate Status Protocol* (OCSP) para consultar a AC em tempo real sobre o *status* de revogação de um certificado.

Há desvantagens nesta abordagens. Primeiramente, se o serviço OCSP está indisponível por qualquer razão, o cliente poderá decidir por aceitar a conexão mesmo sem ter verificado a revogação do certificado do servidor. Segundo, a AC pode ficar sobrecarregada ao responder em tempo real para uma grande quantidade de clientes SSL/TLS, resultando não apenas em atrasos para o cliente, mas também em indisponibilidade do serviço OCSP. O método OCSP *Stapling* resolve estas duas questões.

O OCSP *Stapling* (grampeamento) permite que um servidor (por exemplo, HTTPS) anexe o seu *status* OCSP ao *handshake* do protocolo SSL/TLS. O *status* OCSP do servidor é requisitado pelo próprio servidor à AC emissora, periodicamente, e é assinado pela AC. A Listagem 5.21 mostra como a opção `-status` do comando `s_client` faz com que o cliente solicite explicitamente o *status* do servidor via OCSP *Stapling*. A Listagem 5.21 também mostra a resposta da AC grampeada à resposta do servidor. O restante da saída é omitido.

Listagem 5.21. Teste de grampeamento de OCSP (*OCSP Stapling*).

```
1 C:\>openssl s_client -connect www.exemplo.com:443 -status
2 CONNECTED(000000E8)
3 depth=1 C = US,
      O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assurance Server CA
4 verify error:num=20:unable to get local issuer certificate
5 OCSP response:
6 ======================================
7 OCSP Response Data:
8     OCSP Response Status: successful (0x0)
9     Response Type: Basic OCSP Response
10    Version: 1 (0x0)
11    Responder Id: 5168FF90AF0207753CCCD9656462A212B859723B
12    Produced At: Feb 19 06:26:58 2019 GMT
13    Responses:
14    Certificate ID:
15      Hash Algorithm: sha1
16      Issuer Name Hash: CF26F518FAC97E8F8CB342E01C2F6A109E8E5F0A
17      Issuer Key Hash: 5168FF90AF0207753CCCD9656462A212B859723B
18      Serial Number: 0FDC2551201743C7F56141EC293D66BD
19    Cert Status: good
20    This Update: Feb 19 06:26:58 2019 GMT
21    Next Update: Feb 26 05:41:58 2019 GMT
22
23    Signature Algorithm: sha256WithRSAEncryption
24         0c:54:dd:57:25:1b:55:60:e8:8e:81:29:89:76:4d:8c:9c:f7:
25         33:e4:c9:1a:9f:7e:ae:8b:df:f0:11:87:29:5d:bd:dc:01:5d:
26         87:c9:09:83:f3:c5:b0:d5:4f:68:07:7d:25:7a:54:76:5f:98:
27         d8:af:6e:1e:86:bc:73:6b:20:5d:3f:2e:ad:b8:00:3e:c9:b2:
28         00:db:2f:86:34:24:7a:34:5f:a2:27:27:41:f9:52:6e:bb:b0:
29         07:8a:3e:88:a6:8f:94:61:db:65:31:76:7d:cf:70:91:c2:58:
30         57:c6:34:bf:27:31:2b:2f:3b:6f:84:57:bf:0a:07:42:b2:1b:
31         a2:15:25:e0:35:09:f6:8c:7f:a9:d5:ff:87:d2:88:45:50:46:
32         fb:01:fd:09:da:f6:3a:2c:0a:38:ff:0f:5b:03:9d:77:56:5b:
33         d7:56:63:4a:6e:07:cd:fd:84:f8:0c:af:3f:14:1b:e9:f7:17:
34         d6:cb:c2:d0:3d:0a:99:2d:97:a5:72:15:f2:1e:fe:9d:45:f7:
35         f7:86:7e:56:9e:4b:9a:9a:41:86:f5:f3:9a:43:1e:8a:61:2d:
36         10:4e:cc:96:f1:48:f9:03:f6:c0:2d:a2:62:ca:4c:e9:72:2f:
37         ad:b2:dc:43:f1:d1:84:19:19:48:8c:7c:b3:73:12:cb:0f:66:
38         14:3b:ba:e1
39 ======================================
```

5.5.5. Teste contra o ataque BEAST

O ataque *Browser Exploit Against SSL/TLS* (BEAST) é relativamente antigo (2011) e só afeta principalmente os clientes SSL/TLS nas versões até o TLSv1.0 [Ristic 2011]. Os testes de verificação contra o ataque BEAST no lado servidor ajudam a entender diversas boas práticas de segurança criptográficas implementadas nos servidores web, como, por exemplo, evitar o uso do modo de operação CBC e da encriptação de fluxo com RC4.

O RC4 foi considerado uma alternativa para o CBC no TLSv1.0 na época em que não havia outra encriptação de fluxo disponível para ser usada em vez do RC4. Esta restrição do protocolo levou o público em geral a acreditar que o RC4 era uma solução geral para o BEAST, resultando em substituições inseguras do CBC pelo RC4 em protocolos proprietários [Alkemade 2013]. Por exemplo, o mau uso do RC4 em um protocolo de comunicação segura permitiu a decriptação de mensagens encriptadas trocadas entre dois usuários de um aplicativo de comunicação instantânea [Alkemade 2013]. A vulnerabilidade consistiu na reutilização do fluxo de chaves para encriptar mensagens nas duas direções do canal de comunicação protegido pelo encriptador de fluxo.

Atualmente, a estratégia administrativa recomendada para mitigação do BEAST é adotar apenas as versões mais novas do TLS (1.1, 1.2 e 1.3). Porém, naquelas situações em que é necessário manter o TLSv1.0 para compatibilidade com o legado, apenas uma mitigação parcial é possível. A Listagem 5.22 mostra dois comandos `s_client -connect`. O comando da linha 1 inibe o RC4 e, por isto, só estabelece a conexão se houver alguma criptografia simétrica com CBC para ser usada no TLSv1.0 ou mais baixa. Isto pode ser observado nas linhas 4, 11 e 12. Já o comando da linha 15 oferece o RC4 caso o TLSv1.0 priorize esta mitigação. Como pode ser observado nas linhas 18, 25 e 26, o servidor estabeleceu a conexão vulnerável, indicando que não há mitigação implantada contra o BEAST.

Listagem 5.22. Teste de BEAST.

```
1 C:\>openssl s_client
      -connect exemplo.com:443 -cipher "ALL:!RC4" -no_tls1_1 -no_tls1_2 -no_tls1_3
2
3 [...]
4 New, TLSv1.0, Cipher is ECDHE-RSA-AES128-SHA
5 Server public key is 2048 bit
6 Secure Renegotiation IS supported
7 Compression: NONE
8 Expansion: NONE
9 No ALPN negotiated
10 SSL-Session:
11     Protocol  : TLSv1
12     Cipher    : ECDHE-RSA-AES128-SHA
13 [...]
14
15 C:\>openssl s_client
      -connect exemplo.com:443 -cipher "ALL:+RC4" -no_tls1_1 -no_tls1_2 -no_tls1_3
16
17 [...]
18 New, TLSv1.0, Cipher is ECDHE-RSA-AES128-SHA
19 Server public key is 2048 bit
20 Secure Renegotiation IS supported
21 Compression: NONE
22 Expansion: NONE
23 No ALPN negotiated
24 SSL-Session:
25     Protocol  : TLSv1
26     Cipher    : ECDHE-RSA-AES128-SHA
27 [...]
```

5.5.6. Teste de parâmetros do DH/ECDH

Em versões recentes do `OpenSSL`, o comando `s_client` mostra o tamanho dos parâmetros DH/ECDH usados na conexão recém estabelecida. A Listagem 5.23 mostra três comandos `s_client` com opções `-cipher` diferentes que resultam em parâmetros criptográficos diferentes em cada caso. Nos três casos, as informações de saída localizadas antes e depois dos pontos de interesse foram omitidas por questões de espaço. Estas configurações são suficientemente seguras contra ataques recentes como o LogJam [Log 2016].

No primeiro caso mostrado nas linhas de 1 a 9 da Listagem 5.23, as assinaturas digitais das mensagens do protocolo são geradas com ECDSA sobre SHA256 e as chaves criptográficas foram geradas com tamanho de 253 bits sobre a curva X25519. Já no segundo caso mostrado nas linhas de 11 a 19, as assinaturas digitais das mensagens do protocolo são geradas com RSA sobre SHA512 e as chaves criptográficas ECDH foram geradas com tamanho de 256 bits sobre a curva NIST P-256.

Quando ECDH é utilizado, é importante observar o uso de curvas elípticas fortes e seguras, conforme descrito anteriormente, além de chaves criptográficas suficientemente grandes, com nível de segurança de no mínimo 128 bits (256 bits de tamanho), conforme tabela 5.1. Assim, as curvas X25519 e NIST P-256 são escolhas boas.

Finalmente, no terceiro caso mostrado nas linhas de 21 até 29 da Listagem 5.23, as assinaturas digitais das mensagens do protocolo são geradas com RSA sobre SHA512 e as chaves criptográficas DH foram geradas com tamanho de 2048 bits. Neste caso, vale observar que chaves DH menores de 2048 bits (por exemplo, 512 ou 1024 bits) já são consideradas inseguras para os padrões atuais por que favorecem o ataque LogJam [Log 2016].

Listagem 5.23. Teste de parâmetros de DH/ECDH.

```
1 C:\>openssl s_client -connect www.exemplo.com:443 -cipher ECDHE-ECDSA-AES256-SHA
2 [...]
3 ---
4 No client certificate CA names sent
5 Peer signing digest: SHA256
6 Peer signature type: ECDSA
7 Server Temp Key: X25519, 253 bits
8 ---
9 [...]
10
11 C:\>openssl s_client -connect exemplo.com -port 443 -cipher kECDHE
12 [...]
13 ---
14 No client certificate CA names sent
15 Peer signing digest: SHA512
16 Peer signature type: RSA
17 Server Temp Key: ECDH, P-256, 256 bits
18 ---
19 [...]
20
21 C:\>openssl s_client -connect exemplo.com -port 443 -cipher kDHE
22 [...]
23 ---
24 No client certificate CA names sent
25 Peer signing digest: SHA512
26 Peer signature type: RSA
27 Server Temp Key: DH, 2048 bits
28 ---
29 [...]
```

5.5.7. Testes automático de SSL/TLS com SSLscan

A ferramenta `SSLscan` [rbsec 2019], como toda ferramenta autônoma de varredura de vulnerabilidades, tem a vantagem de poder ser usada para testar um servidor SSL/TLS na rede interna (sem endereço IP público). A Listagem 5.24 mostra o comando `SSLscan` usando para varrer um servidor fictício, onde a opção `-verbose` indica que muitas informações são mostradas na saída. As linhas de 7 até 13 mostram o status de algumas configurações do protocolo, enquanto as linhas de 15 até 18 mostram quais versões são vulneráveis ou não ao Heartbleed [Synopsys 2014]. Em particular, a linha 11 mostra que há suporte à renegociação segura e a linha 13 mostra que a compressão está desabilitada como mitigação contra o ataque CRIME [CRI 2012].

Finalmente, as linhas de 20 até 40 listam as *suites* criptográficas suportadas pelo servidor nas versões TLSv1, TLSv1.1 e TLSv1.2 do protocolo, assim como também a curva elíptica utilizada e o tamanho da chave DHE. As *suites* nas linha 21, 33 e 37 são as preferidas pelo servidor. As informações do certificado foram omitidas.

Listagem 5.24. Ferramenta de teste SSLscan.

```
1 C:\>sslscan -verbose www.exemplo.com:443
2 Version: 1.11.11 Windows 64-bit (Mingw)
3 OpenSSL 1.0.2-chacha (1.0.2g-dev)
4
5 Connected to 74.6.144.137
6
7 Testing SSL server www.exemplo.com on port 443 using SNI name www.exemplo.com
8
9   TLS Fallback SCSV: Server supports TLS Fallback SCSV
10
11  TLS renegotiation: Secure session renegotiation supported
12
13  TLS Compression: Compression disabled
14
15  Heartbleed:
16 TLS 1.2 not vulnerable to heartbleed
17 TLS 1.1 not vulnerable to heartbleed
18 TLS 1.0 not vulnerable to heartbleed
19
20  Supported Server Cipher(s):
21 Preferred TLSv1.2  128 bits  ECDHE-RSA-AES128-GCM-SHA256    Curve P-256 DHE 256
22 Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-GCM-SHA384    Curve P-256 DHE 256
23 Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-SHA256        Curve P-256 DHE 256
24 Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-SHA384        Curve P-256 DHE 256
25 Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-SHA           Curve P-256 DHE 256
26 Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-SHA           Curve P-256 DHE 256
27 Accepted  TLSv1.2  128 bits  AES128-GCM-SHA256
28 Accepted  TLSv1.2  256 bits  AES256-GCM-SHA384
29 Accepted  TLSv1.2  128 bits  AES128-SHA256
30 Accepted  TLSv1.2  256 bits  AES256-SHA256
31 Accepted  TLSv1.2  128 bits  AES128-SHA
32 Accepted  TLSv1.2  256 bits  AES256-SHA
33 Preferred TLSv1.1  128 bits  ECDHE-RSA-AES128-SHA           Curve P-256 DHE 256
34 Accepted  TLSv1.1  256 bits  ECDHE-RSA-AES256-SHA           Curve P-256 DHE 256
35 Accepted  TLSv1.1  128 bits  AES128-SHA
36 Accepted  TLSv1.1  256 bits  AES256-SHA
37 Preferred TLSv1.0  128 bits  ECDHE-RSA-AES128-SHA           Curve P-256 DHE 256
38 Accepted  TLSv1.0  256 bits  ECDHE-RSA-AES256-SHA           Curve P-256 DHE 256
39 Accepted  TLSv1.0  128 bits  AES128-SHA
40 Accepted  TLSv1.0  256 bits  AES256-SHA
41
42  SSL Certificate:
43 Signature Algorithm: sha256WithRSAEncryption
44 RSA Key Strength:    2048
45 [...certificado omitido ...]
```

5.5.8. Testes automáticos de SSL/TLS com TestSSLServer

A ferramenta `TestSSLServer` [Pornin 2017] também tem a vantagem de poder ser usada para testar um servidor SSL/TLS sem endereço IP público. A Listagem 5.25 mostra o comando `TestSSLServer`, onde a opção `-ec` indica que as configurações de curvas elípticas do servidor são mostradas. As linhas de 4 até 10 mostram as *suites* criptográficas preferidas pelo servidor para as versões TLSv1 e TLSv1.1 do protocolo, enquanto as linhas de 11 até 24 mostram as *suites* da versão TLSv1.2 preferidas, onde se pode observar *suites* sem DH/ECDH, apenas com RSA, que não oferecem *forward secrecy*, conforme aviso da linha 51. As informações da cadeia de certificação foram omitidas. As linhas de 28 até 35 mostram o *status* de diversas configurações, enquanto as linhas de 35 até 49 mostram as curvas elípticas suportadas pelo servidor.

Listagem 5.25. Ferramentas de teste TestSSLServer.

```
1 C:\>TestSSLServer2.exe -ec  www.exemplo.com 443
2 Connection: www.exemplo.com:443
3 SNI: www.exemplo.com
4   TLSv1.0:
5       server selection: enforce server preferences
6       3f- (key:  RSA)  ECDHE_RSA_WITH_AES_128_CBC_SHA
7       3f- (key:  RSA)  ECDHE_RSA_WITH_AES_256_CBC_SHA
8       3-- (key:  RSA)  RSA_WITH_AES_128_CBC_SHA
9       3-- (key:  RSA)  RSA_WITH_AES_256_CBC_SHA
10  TLSv1.1: idem
11  TLSv1.2:
12      server selection: enforce server preferences
13      3f- (key:  RSA)  ECDHE_RSA_WITH_AES_128_GCM_SHA256
14      3f- (key:  RSA)  ECDHE_RSA_WITH_AES_256_GCM_SHA384
15      3f- (key:  RSA)  ECDHE_RSA_WITH_AES_128_CBC_SHA256
16      3f- (key:  RSA)  ECDHE_RSA_WITH_AES_256_CBC_SHA384
17      3f- (key:  RSA)  ECDHE_RSA_WITH_AES_128_CBC_SHA
18      3f- (key:  RSA)  ECDHE_RSA_WITH_AES_256_CBC_SHA
19      3-- (key:  RSA)  RSA_WITH_AES_128_GCM_SHA256
20      3-- (key:  RSA)  RSA_WITH_AES_256_GCM_SHA384
21      3-- (key:  RSA)  RSA_WITH_AES_128_CBC_SHA256
22      3-- (key:  RSA)  RSA_WITH_AES_256_CBC_SHA256
23      3-- (key:  RSA)  RSA_WITH_AES_128_CBC_SHA
24      3-- (key:  RSA)  RSA_WITH_AES_256_CBC_SHA
25 ========================================
26 [... cadeia de certificados omitida ... ]
27 ========================================
28 Server compression support: no
29 Server sends a random system time.
30 Secure renegotiation support: yes
31 Encrypt-then-MAC support (RFC 7366): no
32 SSLv2 ClientHello format (for SSLv3+): yes
33 Minimum EC size (no extension):   256
34 Minimum EC size (with extension): 256
35 ECDH parameter reuse:  no
36 Supported curves (size and name) ('*' = selected by server):
37      281  sect283k1 (K-283)
38      282  sect283r1 (B-283)
39      407  sect409k1 (K-409)
40      408  sect409r1 (B-409)
41      569  sect571k1 (K-571)
42      570  sect571r1 (B-571)
43      256  secp256k1
44    * 256  secp256r1 (P-256)
45      384  secp384r1 (P-384)
46      521  secp521r1 (P-521)
47      256  brainpoolP256r1
48      384  brainpoolP384r1
49      512  brainpoolP512r1
50 ========================================
51 WARN[CS006]: Server supports cipher suites with no forward secrecy.
```

5.6. Robustecimento e configuração segura do TLS

Esta seção trata as práticas de configuração segura do protocolo SSL/TLS nos servidores web Apache por meio do robustecimento das configurações do módulo `mod_ssl` [mod_ssl 2019]. A seção também cobre a sintaxe de nomenclatura utilizada para descrição de *suítes* criptográficas no SSL/TLS. O módulo `mod_ssl` é a interface do servidor web Apache para o `OpenSSL`. O site oficial [mod_ssl 2019] tem instruções para *download* e instalação do `mod_ssl` e a sua configuração no Apache. Este texto supõe que o `mod_ssl` já está instalado no servidor Apache, tratando apenas da configuração do módulo.

Aplicando técnicas de auditoria de sistemas, a utilização correta das configurações seguras do SSL/TLS pode ser verificada por meio de inspeções das configurações do `mod_ssl` do servidor web em execução, complementando as técnicas de teste tratadas na seção anterior. Por exemplo, um auditor de sistemas é capaz de verificar a presença do módulo `mod_ssl` em um servidor web Apache ativo (em execução) por meio do comando `httpd.exe -M`, que lista todos os módulos instalados e carregados pelo servidor web. Já que podem existir muitos módulos, um filtro simples seleciona e mostra apenas a linha do `mod_ssl`, se ela existir, como visto a seguir.

```
C:\>httpd.exe -M | find "ssl"
 ssl_module (shared)
```

5.6.1. As *suites* criptográficas das versões TLSv1.2 e TLSv1.3

Uma *suite* criptográfica é um conjunto de funções ou rotinas de criptografia utilizado em uma instância de comunicação segura com o protocolo SSL/TLS. A nomenclatura das *suites* criptográficas segue regras bem definidas que identificam todas as rotinas criptográficas necessárias em um canal de comunicação segura com SSL/TLS. Por exemplo, a Listagem 5.26 mostra algumas *suites* criptográficas disponíveis para o TLsv1.2 que contém ECDHE e AES (linhas de 1 até 13), assim como também para o TLSv1.3 (linhas de 15 até 18), onde, neste último caso, as *suites* são exclusivas da versão 1.3 do protocolo.

Listagem 5.26. Exemplos de *suites* criptográficas nas versões 1.2 e 1.3 do TLS.

```
1  C:\>openssl ciphers -v "TLSv1.2"  | find "ECDHE" | find "AES"
2  ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2  Kx=ECDH   Au=ECDSA  Enc=AESGCM(256)     Mac=AEAD
3  ECDHE-RSA-AES256-GCM-SHA384   TLSv1.2  Kx=ECDH   Au=RSA    Enc=AESGCM(256)     Mac=AEAD
4  ECDHE-ECDSA-AES256-CCM8       TLSv1.2  Kx=ECDH   Au=ECDSA  Enc=AESCCM8(256)    Mac=AEAD
5  ECDHE-ECDSA-AES256-CCM        TLSv1.2  Kx=ECDH   Au=ECDSA  Enc=AESCCM(256)     Mac=AEAD
6  ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2  Kx=ECDH   Au=ECDSA  Enc=AESGCM(128)     Mac=AEAD
7  ECDHE-RSA-AES128-GCM-SHA256   TLSv1.2  Kx=ECDH   Au=RSA    Enc=AESGCM(128)     Mac=AEAD
8  ECDHE-ECDSA-AES128-CCM8       TLSv1.2  Kx=ECDH   Au=ECDSA  Enc=AESCCM8(128)    Mac=AEAD
9  ECDHE-ECDSA-AES128-CCM        TLSv1.2  Kx=ECDH   Au=ECDSA  Enc=AESCCM(128)     Mac=AEAD
10 ECDHE-ECDSA-AES256-SHA384     TLSv1.2  Kx=ECDH   Au=ECDSA  Enc=AES(256)        Mac=SHA384
11 ECDHE-RSA-AES256-SHA384       TLSv1.2  Kx=ECDH   Au=RSA    Enc=AES(256)        Mac=SHA384
12 ECDHE-ECDSA-AES128-SHA256     TLSv1.2  Kx=ECDH   Au=ECDSA  Enc=AES(128)        Mac=SHA256
13 ECDHE-RSA-AES128-SHA256       TLSv1.2  Kx=ECDH   Au=RSA    Enc=AES(128)        Mac=SHA256
14
15 C:\>openssl ciphers -v "TLSv1.3"
16 TLS_AES_256_GCM_SHA384        TLSv1.3 Kx=any  Au=any  Enc=AESGCM(256)             Mac=AEAD
17 TLS_CHACHA20_POLY1305_SHA256  TLSv1.3 Kx=any  Au=any  Enc=CHACHA20/POLY1305(256) Mac=AEAD
18 TLS_AES_128_GCM_SHA256        TLSv1.3 Kx=any  Au=any  Enc=AESGCM(128)             Mac=AEAD
```

Na Listagem 5.26, o comando `openssl ciphers -v "TLSv1.2"` da linha 1 mostra doze (12) *suites* criptográficas da versão TLsv1.2 que usam o acordo de chaves ECDHE

(Diffie-Helman efêmero sobre curvas elípticas) e AES para encriptação simétrica. A análise de cada uma das *suites* desta lista permite identificar os elementos de construção, da esquerda para a direita:

- Criptografia assimétrica para distribuição de chaves, indicando um mecanismo de acordo de chaves ou de transporte de chaves (no exemplo, apenas o ECDHE);
- Esquema de assinaturas digitais para garantia de autenticidade das mensagens do protocolo de *handshake* (no exemplo, RSA PKCS#1v1.5 e ECDSA);
- Encriptação simétrica usada na proteção da confidencialidade dos dados, indicando o nome do algoritmo, o tamanho da chave de sessão e o modo de operação (no exemplo, AES-GCM, AES-CCM, AES-CCM8 ou AES-CBC);
- Mecanismo de MAC ou função de *hash* que compõe o cálculo do HMAC usado na autenticação das mensagens de troca de dados (no exemplo, AEAD, SHA384 ou SHA256).

Já o comando `openssl ciphers -v "TLSv1.3"` da linha 15 mostra apenas três *suites* criptográficas exclusivas para a versão TLSv1.3. Isto ocorre porque nesta versão do protocolo, as opções de criptografia assimétrica para distribuição de chaves (p.ex., ECDHE) e para assinaturas digitais (p.ex., ECDSA) podem ser escolhidas pelo software cliente já no início do *handshake*. Restando, então, apenas a necessidade de negociar com o servidor o mecanismo de encriptação autenticada, com as opções: AES-GCM-128, AES-GCM-256 e CHACHA20-POLY1305-SHA256.

Uma seleção de *suites* criptográficas pode ser escolhida com o uso de palavras chave combinadas com os nomes dos algoritmos criptográficos, formando *strings* de configuração utilizadas por diversas aplicações que incorporam o `OpenSSL`, como é o caso do mod_ssl. Por exemplo, as seguintes *keywords* podem ser usadas tanto no comando `openssl ciphers -v`, quanto na diretiva de configuração `SSLCipherSuite` do módulo `mod_ssl`:

- "DEFAULT":*suites default* da instalação;
- "ALL": todas as *suites*, exceto aquelas com encriptação nula;
- "COMPLEMENTOFDEFAULT": "ALL"menos "DEFAULT";
- "COMPLEMENTOFALL: as *suites* com encriptação nula;
- "HIGH": *suites* com nível de segurança maior que 128 bits;
- "MEDIUM": *suites* com nível de segurança de 128 bits;
- "LOW": *suites* com nível de segurança menor que 128 bits (somente em versões antigas);
- "SSLv3", "TLSv1", "TLSv1.1TLSv1.2", "TLSv1.3": todas as *suites* da versão;
- "SHA", "SHA1", "SHA256", "SHA384": *suites* que usam estes *hashes*;
- "aDH", "aDSS", "aECDH", "aECDSA", "aRSA": *suites* com autenticação;
- "aNULL": *suites* sem autenticação (p.ex., DH anônimo);
- "PSK", "SRP": *suites* com autenticação por chave predefinida ou por senha segura;
- "ADH","AECDH": *suites* inseguras que usam DH anônimo;

- "DH","ECDH": *suites* que usam qualquer DH (anônimo ou efêmero);
- "EDH", "EECDH": *suites* inseguras que usam DH efêmero;
- "kEDH", "kEECDH": *suites* que usam DH efêmero e anônimo;
- "RSA, "kRSA": *suites* que fazem troca de chaves com RSA;
- "AES","AESGCM": *suites* com AES simples ou encriptação autenticada GCM;
- "eNULL","NULL": *suites* sem encriptação.

A *string* de configuração de uma *suite* criptográfica personalizada pode ser montada pela combinação de *keywords*, da esquerda para a direita, separadas por dois pontos (":") ou espaço. O comportamento *default* é sempre adicionar ao final da lista as *suites* associadas à *keyword* a direita. Por exemplo, o comando `openssl ciphers -v "aNULL:eNULL"` seleciona as *suites* sem autenticação e acrescenta à lista as *suites* sem encriptação.

Este comportamento de adicionar *suites* ao final da lista pode ser modificado das seguintes maneiras: um sinal de menos (-") antes da *keyword* elimina as *suites* da lista se elas estiveram lá, mas elas ainda podem ser adicionadas novamente por outra *keyword*, que apareça depois, mais a direita. Já um sinal de exclamação ("!") remove a *suite* definitivamente, enquanto o sinal de mais ("+") move para o final da lista uma *suite* que já está na *string* de configuração. Finalmente, a *keyword* "@STRENGTH"ordena as *suites* da lista pela ordem decrescente do nível de segurança (isto é, a mais segura aparece primeiro, a lista vai da mais segura para a menos segura).

5.6.2. As configurações criptográficas seguras do `mod_ssl`

De acordo com a documentação [Apache Software Foundation 2019], as configurações do servidor Apache (a partir da versão 2.4) estão localizadas nos arquivos `httpd.conf` e `httpd-ssl.conf`. Vale a pena observar a documentação específica da versão utilizada do Apache a fim de determinar a localização dos arquivos de configuração e eventuais mudanças de nomenclatura. O Apache oferece um número grande de configurações em vários aspectos. Este texto trata apenas das configurações criptográficas relacionadas ao `mod_ssl`. Em particular, esta subseção mostra as configurações do `mod_ssl` que utilizam ou afetam a seleção das *suites* criptográficas. Além disso, são analisadas as configurações seguras do `mod_ssl` seguindo boas práticas atualmente em uso [mozilla 2018, SSL Labs 2019]. Primeiramente, são verificadas as configurações que determinam que o módulo foi carregado.

Listagem 5.27. Configurações mínimas do `mod_ssl` no `httpd.conf` do Apache 2.4.x.

```
1 ...
2
3 LoadModule ssl_module modules/mod_ssl.so
4 ...
5
6 # Secure (SSL/TLS) connections
7 Include conf/extra/httpd-ssl.conf
8 ...
9
10 <IfModule ssl_module>
11 SSLRandomSeed startup builtin
12 SSLRandomSeed connect builtin
13 </IfModule>
14 ...
```

A configuração inicial mínima do `mod_ssl` no Apache consiste das seguintes diretivas presentes no arquivo de configuração `httpd.conf` mostradas na Listagem 5.27. As reticências indicam que o arquivo de configuração é longo e muitas linhas foram omitidas. A linha 3 mostra a configuração `LoadModule` para a carga do módulo. Na linha 7, a diretiva `Include` indica que as configurações específicas do módulo estão em um aquivo de configuração separado (`httpd-ssl.conf`) em uma pasta extra. Finalmente, as configurações das linhas de 10 até 13 indicam que o SSL/TLS sempre utilizará sementes pseudo-aleatórias novas tanto na sua inicialização, quanto no estabelecimento de conexões.

A Listagem 5.28 mostra as configurações presentes no arquivo `httpd-ssl.conf` conforme explicado a seguir. A configuração `Listen 443`, na linha 3, indica que o protocolo HTTPS usará a porta 443 (de fato, a porta *default* deste protocolo). As configurações `SSLCipherSuite` e `SSLProxyCipherSuite`, nas linhas 7 e 8, indicam as *suites* criptográficas que serão negociadas pelo servidor Apache diretamente com o software cliente ou por meio de um *proxy* de conexões. A *string* de configuração é uma lista que pode conter *suites* criptográficas, nomes dos algoritmos e as *keywords* explicadas anteriormente. No exemplo da linha 7, primeiro são incluídas as *suites* com nível de segurança maior que 128 bits, depois aquelas com nível de segurança igual a 128 bits. Em seguida, são excluídas todas as *suites* que contenham os algoritmos MD5, RC4 e 3DES.

A configuração `SSLHonorCipherOrder on`, na linha 12, indica que a ordem em que as *suites* e algoritmos criptográficos aparecem na *string* de configuração `SSLCipherSuite` é importante e por isto deve ser honrada na negociação com o cliente SSL/TLS. Por exemplo, a ordem pode indicar que *suites* e algoritmos de melhor desempenho aparecem primeiro e foram escolhidos pelo administrador do servidor web da maneira indicada na *string* de configuração: primeiro as *suites* mais fortes e depois as menos fortes.

As configurações `SSLCompression off` e `SSLSessionTickets off`, nas linhas 16 e 17, indicam que os usos de compressão SSL/TLS e de *tickets* de sessão devem estar desligados. Estas são configurações antigas que, se ativadas, levam a vulnerabilidades conhecidas. Elas mantém a compatibilidade com versões antigas e sistemas legado e, na prática, ficam sempre desativadas em versões novas.

As configurações `SSLProtocol` e `SSLProxyProtocol`, nas linhas 21 e 22, informam quais versões do SSL/TLS são habilitadas no servidor. Geralmente, deseja-se todas as versões recentes, excluindo-se as versões antigas e inseguras (por exemplo, `All -SSLv2 -SSLv3 -TLSv1`). Versões mais recentes do `OpenSSL` já não oferecem a opção de configuração `-SSLv2`, uma vez que não possuem mais esta versão antiga do protocolo. Vale ainda observar que a configuração `SSLProtocol` mostrada segue o princípio da lista de exclusão (lista negativa). Outra opção é adotar uma lista de inclusão (lista positiva) que contenha explicitamente apenas as versões desejadas. Por exemplo, `TLSv1.1 TLSv1.2 TLSv1.3`).

As configurações relacionadas ao *cache* de sessão SSL/TLS, nas linhas 26 e 27, indicam o local de armazenamento e o tempo limite de duração da sessão. Já as configurações relacionadas ao OCSP *Stapling*, nas linhas de 31 até 34, indicam que o *Stapling* está ativado, onde o *cache* de OCSP está armazenado, e os tempos limite de erro e de permanência de uma resposta OSCP no *cache*.

No arquivo `httpd-ssl.conf`, o bloco ou contexto `<VirtualHost>` atende à

necessidade de compartimentação de aplicações no servidor Apache e contém configurações específicas das aplicações indicadas nos contextos. A Listagem 5.28 mostra o contexto *default* nas linhas de 38 até 49 que contém as configurações globais do servidor web (`<VirtualHost _default_:443>` ou `<VirtualHost *:443>`). Configurações específicas reproduzem este bloco de configuração e substituem o asterisco (*) pelo nome de domínio ou endereço IP do servidor virtual. Quando está presente no contexto `<VirtualHost>`, a configuração `SSLEngine On | Off` ativa ou desativa o SSL/TLS no contexto em questão. Já as configurações `SSLCertificateFile` e `SSLCertificateKeyFile` informam onde o Apache vai achar o certificado digital do virtual host e a chave privada correspondente.

Listagem 5.28. Configurações do `mod_ssl` no arquivo `httpd-ssl.conf` do Apache 2.4.

```
1
2 # qual a porta usada para o https
3 Listen 443
4 ...
5
6 # string configura as suites criptograficas com keywords e nomes do OpenSSL
7 SSLCipherSuite HIGH:MEDIUM:!MD5:!RC4:!3DES
8 SSLProxyCipherSuite HIGH:MEDIUM:!MD5:!RC4:!3DES
9 ...
10
11 # honra a ordem em que as suites sao listadas (motivo eh desempenho).
12 SSLHonorCipherOrder on
13 ...
14
15 # desabilta compression e session tickets
16 SSLCompression          off
17 SSLSessionTickets       off
18
19 ...
20 #permite TLsv1.1, TLSv1.2 e TLSv1.3 mas nao permite SSLv2, SSLv3 and TLsv1
21 SSLProtocol All -SSLv2 -SSLv3 -TLSv1
22 SSLProxyProtocol All -SSLv2 -SSLv3 -TLSv1
23 ...
24
25 #Cache da sessao (reutilizacao de parametros para favorecer desempenho)
26 SSLSessionCache         "shmcb:${SRVROOT}/logs/ssl_scache(512000)"
27 SSLSessionCacheTimeout  300
28 ...
29
30 # OCSP Stapling (desabilitado por default)
31 SSLUseStapling On
32 SSLStaplingCache "shmcb:${SRVROOT}/logs/ssl_stapling(32768)"
33 SSLStaplingStandardCacheTimeout 3600
34 SSLStaplingErrorCacheTimeout 600
35 ...
36
37 # Contexto do virtual host default
38 <VirtualHost _default_:443>
39   ...
40   # Habilita ou desabilita o SSL no virtual host
41   SSLEngine On
42
43   # Caminho para o certificado do servidor
44   SSLCertificateFile /usr/local/apache/conf/ssl/server.crt
45
46   # Caminho para a chave privada do servidor
47   SSLCertificateKeyFile /usr/local/apache/conf/ssl/server.key
48 ...
49 </VirtualHost>
50 ...
```

5.7. Considerações finais

Este texto descreve bons usos da criptografia no protocolo SSL/TLS, contribuindo para a administração de redes e de sistemas, facilitando o aprofundamento em estudos futuros. Em particular, profissionais atuando em redes de computadores e sistemas distribuídos são instruídos sobre SSL/TLS por meio de um tutorial `OpenSSL`, análises das configurações no `mod_ssl` e testes de segurança no servidor web Apache, O texto aborda a versão nova do SSL/TLS, ainda em processo de adoção pela indústria. Por isto, contribui para que administradores de TI entendam os benefícios da nova versão e avancem na atualização de seus sistemas.

Há oportunidades de pesquisa e desenvolvimento em SSL/TLS onde há escassez de especialistas em criptografia, e as configurações de SSL/TLS são feitas manualmente por profissionais de outras áreas, geralmente sobrecarregados com outras atividades. Três áreas em particular chamam a atenção: (*i*) a geração automática de configurações seguras, (*ii*) a verificação automática da segurança do TLS e (*iii*) a correção automática de configurações inseguras.

Profissionais de TI precisam de bons exemplos de configurações seguras para seguir atuando. Neste sentido, geradores de modelos de configuração seguras (atualizados para as versões novas e em diversas tecnologias) contribuem bastante nesta direção. Bons exemplos facilitam o aprendizado, servem de base para personalizações e aceleram as correções de defeitos. Por outro lado, maus exemplos (explicitamente documentados como tal e explicados) são uma ferramenta poderosa de conscientização sobre como evitar configurações inseguras.

A utilização de ferramentas de varredura de vulnerabilidades em implantações do SSL/-TLS é uma área já bastante ativa com uma variedade grande de ferramentas disponíveis. Os desafios tecnológicos desta área em particular estão não apenas na capacidade das ferramentas em diminuir a ocorrência de falsos positivos e de falsos negativos, mas principalmente na capacidade de comunicar suas descobertas de maneira adequada para não especialistas. Por isto, novas ferramentas de testes, fáceis de usar corretamente, atualizadas e amplamente disponíveis, também contribuem para a conscientização dos profissionais de TI em relação ao estado de suas infraestruturas, sendo um passo necessário na atualização de infraestruturas existentes.

Finalmente, no que se refere à detecção e correção automáticas de configurações inseguras, infraestruturas auto-adaptativas podem se beneficiar da detecção de configurações inseguras e da sua correção automática e transparente. Por exemplo, na implantação de contêineres de aplicação em infraestruturas de nuvem, um contêiner com um Apache configurado de modo inseguro para SSL/TLS poderia ser corrigido por edições controladas e automáticas de seus arquivos de configuração, em vez de ser rejeitado sumariamente.

Agradecimentos

Este trabalho foi realizado no Laboratório de Segurança e Criptografia (LASCA) do Instituto de Computação, Universidade Estadual de Campinas. Os autores agradecem o apoio financeiro do projeto ATMOSPHERE (**A**daptive, **T**rustworthy, **M**anageable, **O**rchestrated, **S**ecure, **P**rivacy-assuring, **H**ybrid, **E**cosystem for **RE**silient Cloud Computing), RNP e MCTIC, no âmbito to acordo de cooperação Número 51119. *ATMOSPHERE is funded by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement No 777154*. Agradecem também o apoio da Fapesp, por meio do projeto temático *Segurança e Confiabilidade da Informação: Teoria e Aplicações*, no. 2013/25.977-7.

Referências

[CRI 2012] (2012). The CRIME attack: netifera.com. *URL: http://netifera.com*.

[BRE 2013] (2013). The BREACH attack: SSL GONE IN 30 SECONDS. *URL: http://breachattack.com*.

[Log 2016] (2016). The logjam attack and weak diffie-hellman. *URL: https://weakdh.org*.

[Adrian et al. 2015] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., and Others (2015). Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17. ACM.

[AlFardan et al. 2013] AlFardan, N. J., Bernstein, D. J., Paterson, K. G., Poettering, B., and Schuldt, J. C. N. (2013). On the security of rc4 in tls. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 305–320, Berkeley, CA, USA. USENIX Association.

[Alkemade 2013] Alkemade, T. (2013). Mitigating the beast attack on tls. *URL: https://blog.thijsalkema.de/blog/2013/10/08/piercing-through-whatsapp-s-encryption*.

[Apache Software Foundation 2019] Apache Software Foundation (2019). Apache HTTP Server Project. *URL: http://httpd.apache.org*.

[Aviram et al. 2016] Aviram, N., Schinzel, S., Somorovsky, J., Heninger, N., Dankel, M., Steube, J., Valenta, L., Adrian, D., Halderman, J. A., Dukhovni, V., Käsper, E., Cohney, S., Engels, S., Paar, C., and Shavitt, Y. (2016). DROWN: Breaking TLS using SSLv2. *Proceedings of the 25th USENIX Security Symposium*, (August):1–18.

[Bernstein 2006] Bernstein, D. J. (2006). Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer.

[Bernstein et al. 2013] Bernstein, D. J., Lange, T., et al. (2013). Safecurves: choosing safe curves for elliptic-curve cryptography. *URL: http://safecurves.cr.yp.to*.

[Bleichenbacher 1998] Bleichenbacher, D. (1998). Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs# 1. In *Annual International Cryptology Conference*, pages 1–12. Springer.

[BlueKrypt] BlueKrypt. Cryptographic Key Length Recommendation. *URL: http://www.keylength.com*.

[Braga and Dahab 2015] Braga, A. and Dahab, R. (2015). Introdução à Criptografia para Programadores: Evitando Maus Usos da Criptografia em Sistemas de Software. In *Caderno de minicursos do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg 2015*, pages 1–50. Sociedade Brasileira de Computação.

[Braga and Dahab 2018] Braga, A. and Dahab, R. (2018). Criptografia assimétrica para programadores - evitando outros maus usos da criptografia em sistemas de software. In *Caderno de minicursos do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg 2018*, pages 1–50. Sociedade Brasileira de Computação.

[Calvo 2018] Calvo, R. (2018). The true cost of certificate authority trials: Can you trust them? *URL: https://www.isc2.org/News-and-Events/Infosecurity-Professional-Insights*.

[Chandra et al. 2002] Chandra, P., Messier, M., and Viega, J. (2002). Network security with OpenSSL. *O'Reily*.

[Diffie and Hellman 1976] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654.

[Eldewahi et al. 2015] Eldewahi, A. E. W., Sharfi, T. M. H., Mansor, A. A., Mohamed, N. A. F., and Alwahbani, S. M. H. (2015). Ssl/tls attacks: Analysis and evaluation. In *2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*, pages 203–208.

[Fardan and Paterson 2013] Fardan, N. A. and Paterson, K. (2013). Lucky thirteen: Breaking the TLS and DTLS record protocols. *Security and Privacy (SP), 2013 IEEE Symposium on (2013)*.

[Gluck et al. 2013] Gluck, Y., Harris, N., and Angel, A. (2013). BREACH: Reviving the CRIME Attack. In *Black Hat Conference*.

[Hankerson et al. 2004] Hankerson, D., Vanstone, S., and Menezes, A. (2004). *Guide to elliptic curve cryptography*.

[Jonsson and Burt Kaliski 2003] Jonsson, J. and Burt Kaliski (2003). RSA Laboratories Public-Key Cryptography Standards (PKCS)#1: RSA Cryptography Specifications Version 2.1. *URL: https://tools.ietf.org/html/rfc3447*.

[Koblitz 1987] Koblitz, N. (1987). Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209.

[Miller 1985] Miller, V. S. (1985). Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer.

[mod_ssl 2019] mod_ssl (2019). mod_ssl:the apache interface to openssl. *URL: http://www.modssl.org*.

[Möller et al. 2014] Möller, B., Duong, T., and Kotowicz, K. (2014). The POODLE attack. *URL: https://www.openssl.org/ bodo/ssl-poodle.pdf*.

[mozilla 2018] mozilla (2018). Mozilla's server side tls guidelines. *URL: https://wiki.mozilla.org/Security/Server_Side_TLS*.

[mozilla 2019] mozilla (2019). Http and ssl observatory. *URL: https://observatory.mozilla.org*.

[NIST 2001] NIST (2001). Recommendation for block cipher modes of operation. *NIST SP 800-38A URL: http:csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf*.

[NIST 2007] NIST (2007). Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. *NIST SP 800-38D URL: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf*.

[NIST 2012] NIST (2012). Recommendation for Key Management – Part 1: General (Revision 3). *URL: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf*.

[NIST 2013] NIST (2013). Digital Signature Standard (DSS).

[OpenSSL.org] OpenSSL.org. OpenSSL Cryptography and SSL/TLS toolkit. *URL: OpenSSL.org*.

[OWASP 2015] OWASP (2015). OWASP Testing Project v4. *URL: https://www.owasp.org/index.php/OWASP_Testing_Project*.

[Pornin 2017] Pornin, T. (2017). Testsslserver. *URL: https://www.bolet.org/TestSSLServer*.

[Qualys 2019] Qualys, I. (2019). Qualys ssl labs. *URL: https://www.ssllabs.com*.

[rbsec 2019] rbsec (2019). Sslscan. *URL: https://github.com/rbsec/sslscan*.

[Rescorla 2018] Rescorla, E. (2018). The transport layer security (tls) protocol version 1.3. *URL: https://tools.ietf.org/html/rfc8446*.

[Ristic 2011] Ristic, I. (2011). Mitigating the beast attack on tls. *URL: https://blog.qualys.com/ssllabs/2011/10/17/mitigating-the-beast-attack-on-tls*.

[Ristic 2015] Ristic, I. (2015). *OpenSSL cookbook (Version 2.1-draft, June 2018)*. Feisty Duck, 2nd. edition.

[Rivest et al. 1978] Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

[SEC 2000] SEC, S. (2000). Sec 2: Recommended elliptic curve domain parameters, version 1. *Standards for Efficient Cryptography Group, Certicom Corp (http://www.secg.org/)*.

[SEC 2010] SEC, S. (2010). Sec 2: Recommended elliptic curve domain parameters, version 2. *Standards for Efficient Cryptography Group, Certicom Corp ()http://www.secg.org/*.

[SSL Labs 2019] SSL Labs (2019). Ssl and tls deployment best practices. *URL: https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices*.

[Stallings 2003] Stallings, W. (2003). *Cryptography and network security, principles and practices*.

[Sullivan 2018] Sullivan, N. (2018). A detailed look at rfc 8446 (a.k.a. tls 1.3). *URL: https://blog.cloudflare.com/rfc-8446-aka-tls-1-3*.

[Synopsys 2014] Synopsys (2014). The Heartbleed Bug. *URL: http://heartbleed.com*.

[Wikipedia 2018] Wikipedia (2018). Netscape. *URL: https://en.wikipedia.org/wiki/Netscape*.

Capítulo

6

Introdução à Ciência de Dados: Uma Visão Pragmática utilizando Python, Aplicações e Oportunidades em Redes de Computadores

Giovanni Comarela (UFV), Gabriel Franco (UFV), Celio Trois (UFSM), Alextian Liberato (IFES), Magnos Martinello (UFES), João Henrique Corrêa (UFES) e Rodolfo Villaça (UFES)

Abstract

Internet of Things, smart cities, crowdsourcing, and other neotechnologies are making the world and people more and more connected, causing colossal amounts of information to be transmitted through the network. In order to extract new knowledge and relevant information from these data, Data Science*, which covers several disciplines, such as statistics, data mining, machine learning, and artificial intelligence, is becoming popular and gaining space in the most diverse areas of knowledge. In this context, this course presents, in a pragmatic way, through the Python language and specific libraries, the methods most used by data scientists to manipulate and analyze information, applying them in the resolution of relevant problems in the area of computer networks.*

Resumo

Internet das Coisas, cidades inteligentes, crowdsourcing e outros neotecnologismos estão tornando o mundo e as pessoas cada vez mais conectados, fazendo que quantidades colossais de informação sejam transmitidas pela rede. No intuito de extrair novos conhecimentos e informações relevantes desses dados, a Ciência de Dados*, que abrange várias disciplinas, tais como estatística, mineração de dados, aprendizado de máquina e inteligência artificial, está se popularizando e ganhando espaço nas mais diversas áreas de conhecimento. Nesse contexto, esse minicurso apresenta, de forma pragmática, através da linguagem Python e bibliotecas específicas, os métodos mais usados para manipular e analisar dados e extrair informações, aplicando-os na resolução de problemas relevantes da área de redes de computadores.*

Introdução

Recentemente o termo "Dataist" foi cunhado no vale do silício como uma espécie de religião dos dados. O objetivo é maximizar o fluxo de dados conectando mais e mais mídias, produzindo e consumindo cada vez mais informação; os profetas do "Dataism" conscientemente usam a linguagem tradicional messiânica[1]. Como toda a religião de sucesso, o "Dataism" é também missionário e tem como seu mandamento ligar tudo ao sistema, incluindo os hereges. E tudo significa mais do que apenas conectar humanos, isso quer dizer conectar todas as coisas. Os carros nas ruas, refrigeradores nas cozinhas, e árvores nas florestas - tudo deve estar conectado à Internet-de-todas-as-coisas. Carros vão se comunicar entre si, as árvores na selva vão relatar os níveis de dióxido de carbono. Em tese, não deve-se deixar qualquer parte do universo desconectado da grande rede.

O "Dataism" [Harari, 2016] apoia a liberdade de informação como o maior bem de todos. Entretanto, "Dataists" acreditam que humanos não conseguem lidar com o imenso fluxo de dados, e não distinguem mais dados de informação. O trabalho de processamento de dados deveria assim ser passado aos algoritmos, cuja capacidade excede de longe o cérebro humano. Na prática, isso significa que os "Dataists" são céticos em relação ao processamento de grandes volumes de dados pelo cérebro humano, e preferem confiar em algoritmos. Como exemplo da efetividade dos algoritmos, o Facebook é capaz de apontar com precisão a intenção de votos nas eleições, identificar os eleitores que ainda não se decidiram e determinar o que cada candidato precisa dizer para conquistar mais votos [2].

Como obter esses dados como a matéria-prima dourada? Muitas pessoas proveem dados gratuitamente para ter acesso aos serviços oferecidos na Internet. De fato, a popularidade atingida pela Internet e pela Web nas últimas décadas faz com que uma grande quantidade de dados seja gerada diariamente. Esses dados podem estar relacionados tanto aos usuários da rede (e.g., páginas Web visitadas, produtos comprados, pesquisas em máquinas de busca, amizades em redes sociais e filmes assistidos) quanto a própria rede em si (e.g., utilização e topologia da rede, falhas em roteadores e *links*, disponibilidade e desempenho de serviços e dinâmica de roteamento). De qualquer forma, analisar essa quantidade massiva de dados é essencial para que empresas possam oferecer melhores serviços e, consequentemente, possam sobreviver em um mercado que é tão competitivo.

Esta crescente necessidade de analisar dados fez surgir a demanda por profissionais bem versados em estatística, aprendizado de máquina, mineração de dados, inteligência artificial, econometria e várias outras áreas correlatas. Informalmente[3], tal profissional é referido como *cientista de dados* e a área que trabalha, *ciência de dados*.

A abordagem adotada nesse texto é apresentar um resumo de diversas competências básicas de um profissional da área Ciência de Dados. Outro aspecto considerado é reunir alguns exemplos de trabalhos científicos que mostram a importância da análise de dados em redes de computadores. A partir de exemplos, pretende-se motivar o leitor sobre a importância do tema para a área de redes, para então mostrar que, apesar de complexas, muitas técnicas podem ser utilizadas de forma simples por meio da linguagem de

[1] Por exemplo, no livro de profecias de "The Singularity is Near [Kurzweil, 2010]".

[2] https://bdtechtalks.com/2018/01/31/yuval-harari-wef-ai-big-data-digital-dictatorship/

[3] Apesar de esforços, ainda não há um consenso na academia e indústria, nem mesmo regulamentação governamental, sobre a profissão cientista de dados.

programação Python e várias de suas bibliotecas.

Há bastante tempo, a comunidade acadêmica de redes vem estudando técnicas de manipulação e análise de dados de forma a extrair conhecimento para apoiar a tomada de decisão [Boutaba et al., 2018]. Mais especificamente no contexto do Simpósio Brasileiro de Redes de Computadores, minicursos relacionados, mas não equivalentes, ao tema deste texto e à Ciência de Dados já foram apresentados. Por exemplo, em [Carvalho et al., 2014] foi explorado o advento da e-Ciência e aplicações de *Big Data*, que trouxe consigo a necessidade de se trabalhar com volumes cada vez maiores de dados oriundos de sensores, equipamentos especializados e centros de computação de alto desempenho. Este trabalho visou apresentar os desafios impostos pelas aplicações de e-Ciência com o foco em um modelo de rede denominado DMZ Científica, bem como as tecnologias existentes e futuras para a sua implantação e utilização. Outro minicurso ministrado no contexto de análise de dados foi [Celes et al., 2017], intitulado "Big Data Analytics no Projeto de Redes Móveis: Modelos, Protocolos e Aplicações". Este, por sua vez, investigou a extração de conhecimento a partir dados com informações espaço-temporais de diferentes entidades, tais como pessoas, veículos e objetos no domínio de redes móveis. Nesse sentido, o minicurso proposto usou dados históricos dos usuários para obter características importantes que permitiram detectar padrões e realizar predições, tanto da mobilidade desses usuários como do tráfego de dados no âmbito espaço-temporal.

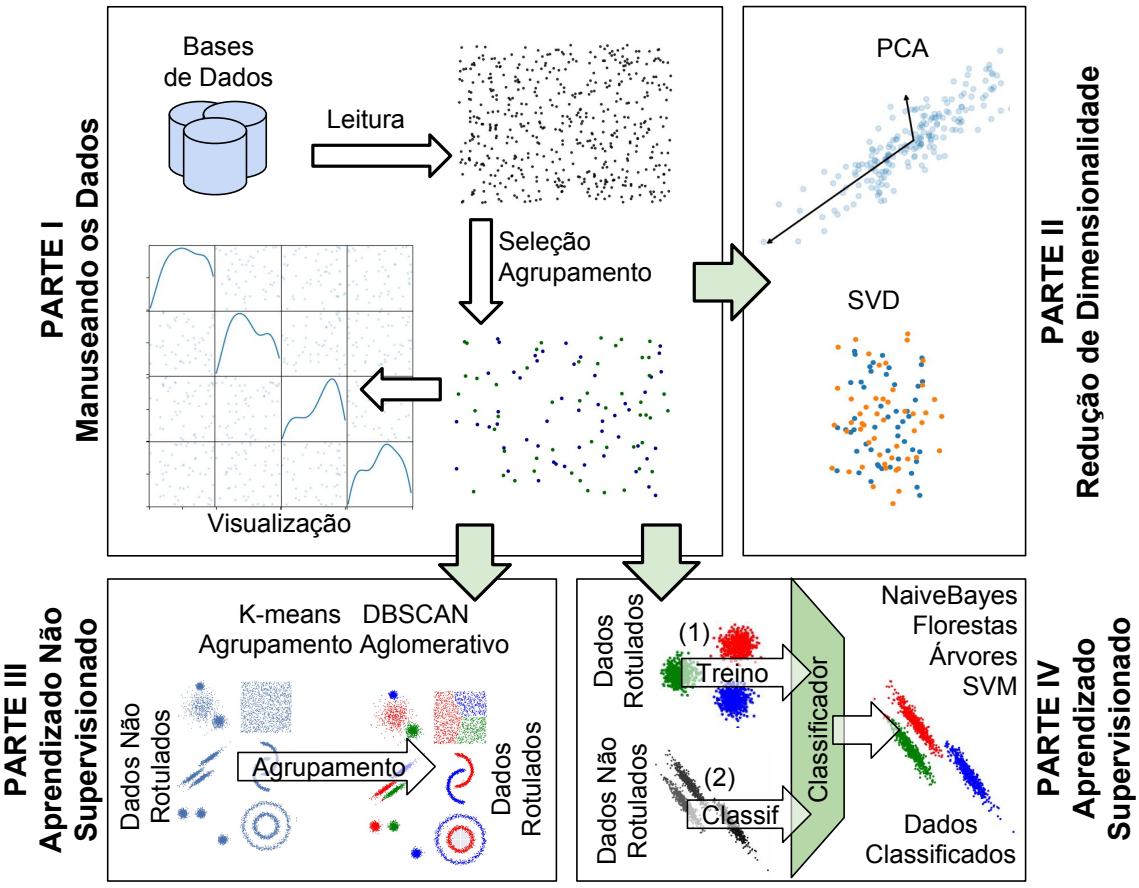


Figura 6.1: Visão geral dos tópicos abordados no minicurso.

O conteúdo deste texto está dividido em quatro partes principais, conforme apresentado na Figura 6.1 e explicado a seguir:

- A Parte I apresenta recursos para **leitura e manipulação de dados** por meio da biblioteca *Pandas* do Python.
- Na Parte II, mostra-se como efetuar **redução de dimensionalidade** via análise de componentes principais e decomposição de valores singulares. Os conceitos apresentados são então utilizados para mostrar como é possível detectar anomalias em matrizes de tráfego, assim como proposto em [Lakhina et al., 2005].
- A Parte III parte tem foco em **aprendizado não supervisionado**, onde são apresentados diferentes paradigmas para realizar agrupamento de dados: K-Means, agrupamento aglomerativo, agrupamento baseado em densidade e agrupamento espectral. Além disso, mostra-se, como exemplo de aplicação, como algumas dessas técnicas podem ser utilizadas para detectar comunidades em redes.
- A parte IV tem como objetivo introduzir algumas **técnicas de classificação**, i.e., aprendizado supervisionado, mostrando, além dos principais classificadores, **metodologias para treino, avaliação, validação e teste** de modelos. Nessa parte, apresenta-se um exemplo referente às comunicações de aplicações científicas de alto desempenho em um centro de dados. São analisadas as imagens das matrizes de tráfego de aplicações científicas, visando identificar, através de aprendizado supervisionado, qual aplicação está sendo executada [Trois et al., 2016, Trois et al., 2018].

PARTE I - Leitura e Manipulação de dados

O primeiro passo para que dados sejam analisados é a sua aquisição. Muitas vezes os dados lidos contém valores indesejáveis, sendo necessário efetuar operações tais como seleção, ordenação e agrupamento, antes que a análise propriamente dita seja feita. Então, nessa parte do minicurso serão apresentadas algumas funcionalidades da biblioteca *Pandas*[4] do Python, que auxilia na leitura e manipulação de dados. Os exemplos citados são baseados em [McKinney, 2013] e na documentação da biblioteca.

DataFrame

O conceito mais importante da biblioteca *Pandas* é o de `DataFrame`, que consiste em uma estrutura bi-dimensional com elementos rotulados. Os rótulos das linhas estão em um atributo `index`, e os das colunas em `columns`. Essa estrutura pode ser abstraída como uma planilha de dados ou como uma tabela de um banco de dados relacional.

Um `DataFrame` pode ser criado de várias formas. Por exemplo, é possível inicializá-lo com os valores de uma matriz ou importá-lo de um arquivo de extensão `.csv`. O primeiro caso é ilustrado no código a seguir.

[4] `https://pandas.pydata.org`

```
In [1]: import numpy as np
In [2]: import pandas as pd
In [3]: dates = pd.date_range('20130101', periods=6)
In [4]: df = pd.DataFrame(np.random.randn(6,4), index=dates,
   ....:                                  columns=list('ABCD'))
In [5]: df
Out[5]:
                   A         B         C         D
2013-01-01 -0.684768  0.857024  0.040317 -2.954732
2013-01-02  1.015421  0.182357  0.483999 -0.649455
2013-01-03  0.212769 -1.153668 -1.006612 -0.642244
2013-01-04 -0.683321  0.297379  0.693017  0.021572
2013-01-05  0.376206  0.054179 -0.373296 -0.907636
2013-01-06  1.416673 -0.825072  1.505570 -2.277256
```

Para importar um `DataFrame` de um arquivo de extensão `.csv`, a função `read_csv` é utilizada.

```
In [6]: df = pd.read_csv('example.csv')
In [7]: df
Out[7]:
        foo       bar
0  1.274366  0.593544
1 -0.409548  0.999609
```

Se for necessário utilizar alguma coluna do arquivo como `index`, é preciso utilizar o parâmetro `index_col`, passando a posição da coluna que será utilizada.

```
In [8]: df = pd.read_csv('example.csv', index_col=0)
In [9]: df
Out[10]:
                bar
foo
 1.274366  0.593544
-0.409548  0.999609
```

Manipulação Básica

Há varias formas de se manimpular um `DataFrame` de forma simples e eficiente. Alguns exemplos são dados a seguir.

Abaixo, um novo `DataFrame` é criado passando uma matriz de tamanho 8×4, criada aleatoriamente, com os índices sendo dados pelo arranjo de datas e as colunas por uma lista de caracteres.

```
In [10]: dates = pd.date_range('20130101', periods=8)
In [11]: df = pd.DataFrame(np.random.randn(8,4), index=dates,
   ....:                                  columns=list('ABCD'))
```

É possível ver apenas as 5 primeiras (últimas) linhas de um `DataFrame` com o método `head` (`tail`).

```
In [12]: df.head()
Out[12]:
                   A         B         C         D
2013-01-01  1.040374  0.264662 -0.906420 -1.368221
2013-01-02 -0.588482 -1.739829  1.581274 -0.488375
2013-01-03  0.116974 -0.550166  0.489298 -1.298768
2013-01-04 -0.301473 -0.373992  0.132532  1.560100
2013-01-05 -1.713864  0.949441 -1.035164  2.080529
```

A matriz de valores correspondente ao `DataFrame` pode ser acessada pelo atributo `values`.

```
In [14]: df.values
Out[14]:
array([[ 1.04037406,  0.26466216, -0.90642008, -1.36822133],
       [-0.58848233, -1.73982906,  1.58127391, -0.48837542],
       [ 0.11697361, -0.55016634,  0.48929796, -1.29876785],
       [-0.30147253, -0.37399189,  0.13253181,  1.56009999],
       [-1.71386375,  0.94944098, -1.03516444,  2.0805291 ],
       [ 0.70227858,  0.05009946, -0.91479705,  0.44931671],
       [ 0.28091816,  1.76048814, -1.46628267,  1.03673774],
       [-0.22356422, -0.31369806, -1.57005912,  0.48789771]])
```

O `DataFrame` transposto é obtido pelo atributo `T`.

```
In [15]: df.T
Out[15]:
   2013-01-01  2013-01-02     ...      2013-01-07  2013-01-08
A    1.040374   -0.588482     ...        0.280918   -0.223564
B    0.264662   -1.739829     ...        1.760488   -0.313698
C   -0.906420    1.581274     ...       -1.466283   -1.570059
D   -1.368221   -0.488375     ...        1.036738    0.487898
[4 rows x 8 columns]
```

Existem outros métodos mais sofisticados para descrever e manipular os dados. Um deles é o `describe`, que gera estatísticas descritivas sobre os dados, excluindo atributos não numéricos.

```
In [16]: df.describe()
Out[16]:
              A         B         C         D
count  8.000000  8.000000  8.000000  8.000000
mean  -0.002739  0.374372  0.319950  0.317294
std    1.148546  0.695346  0.926048  1.161585
min   -1.245660 -1.033110 -0.999356 -1.458622
25%   -1.208835  0.237243 -0.367379 -0.257647
50%    0.150425  0.473167  0.686695  0.173904
75%    0.721654  0.758711  0.945877  1.009124
max    1.746989  1.142517  1.250873  2.285539
```

É possível ordenar um `DataFrame` pelos valores dos índices com o método `sort_index`. Neste caso, de maneira decrescente, utilizando o parâmetro `ascending`.

```
In [17]: df.sort_index(axis = 0, ascending = False)
Out[17]:
                   A         B         C         D
2013-01-08  0.961354 -0.130800  0.542649  1.199216
2013-01-07  0.539913  1.047744 -0.157967 -0.656732
2013-01-06 -1.245660  1.142517  0.830741  2.285539
2013-01-05 -1.223127  0.662367  0.847614  0.945761
2013-01-04 -0.239063  0.359925 -0.999356 -0.124618
2013-01-03 -1.204071 -1.033110 -0.995615 -1.458622
2013-01-02  1.746989  0.440088  1.250873  0.001847
2013-01-01  0.641754  0.506245  1.240665  0.345961
```

O `DataFrame` oferece um método para ordenar por valores, chamado `sort_values`. Neste caso, é necessário passar uma lista de colunas para ser feita a ordenação. Os valores são ordenados pela primeira coluna da lista. Se houver outra coluna, os valores são ordenados por esta, mas respeitando a ordenação da primeira coluna, e assim sucessivamente. Para exemplificar esse método, abaixo é criado um `DataFrame` de exemplo passando um dicionário para o construtor.

```
In [18]: df = pd.DataFrame({
   ....:       'col1' : ['A', 'A', 'B', np.nan, 'D', 'C'],
   ....:       'col2' : [2, 1, 9, 8, 7, 4],
   ....:       'col3': [0, 1, 9, 4, 2, 3],
   ....:   })
In [19]: df
Out[19]:
  col1  col2  col3
0    A     2     0
1    A     1     1
2    B     9     9
3  NaN     8     4
4    D     7     2
5    C     4     3
```

O `DataFrame` é ordenado pelos valores da primeira coluna.

```
In [20]: df.sort_values(by = ['col1'])
Out[20]:
  col1  col2  col3
0    A     2     0
1    A     1     1
2    B     9     9
5    C     4     3
4    D     7     2
3  NaN     8     4
```

O mesmo `DataFrame` é ordenado pelos valores da primeira coluna, e depois pelos valores da segunda coluna respeitando a ordenação feita pela primeira coluna.

```
In [21]: df.sort_values(by = ['col1', 'col2'])
Out[21]:
```

```
  col1  col2  col3
1    A     1     1
0    A     2     0
2    B     9     9
5    C     4     3
4    D     7     2
3  NaN     8     4
```

Seleção em DataFrames

Há várias formas de selecionar dados em um `DataFrame`. É permitido selecionar áreas de interesse por índices ou colunas de forma direta.

Um novo `DataFrame` é criado passando uma matriz de tamanho 6×4 criada aleatoriamente, com os índices sendo dados pelo arranjo de datas e as colunas por uma lista de caracteres.

```
In [22]: dates = pd.date_range('20130101', periods=6)
In [23]: df = pd.DataFrame(np.random.randn(6,4), index=dates,
   ....:                                columns=list('ABCD'))
In [24]: df
Out[24]:
                   A         B         C         D
2013-01-01  1.795895  0.428990 -0.911703  1.536664
2013-01-02  0.729177  1.448289 -0.302455 -0.079658
2013-01-03  0.276477 -1.167711 -0.788572 -0.609279
2013-01-04  0.088234  0.607176  1.080679 -1.163709
2013-01-05 -1.780621  0.293291  0.727409  1.817065
2013-01-06 -0.499000 -2.246464 -0.911982 -0.118192
```

Para acessar o valor de uma coluna é utilizado o operador `[]` ou o atributo com o nome da coluna.

```
In [25]: df['A'] #Ou df.A
Out[25]:
2013-01-01    1.795895
2013-01-02    0.729177
2013-01-03    0.276477
2013-01-04    0.088234
2013-01-05   -1.780621
2013-01-06   -0.499000
Freq: D, Name: A, dtype: float64
```

Para selecionar linhas pelo índice, é possível utilizar o operador `[:]`. São selecionadas as três primeiras linhas do `DataFrame`.

```
In [26]: df[0:3]
Out[26]:
                   A         B         C         D
2013-01-01  1.795895  0.428990 -0.911703  1.536664
2013-01-02  0.729177  1.448289 -0.302455 -0.079658
2013-01-03  0.276477 -1.167711 -0.788572 -0.609279
```

As linhas também podem ser selecionadas por seus rótulos. Abaixo, são selecionadas as linhas do índice `20130102` ao `20130104`.

```
In [27]: df['20130102':'20130104']
Out[27]:
                   A         B         C         D
2013-01-02  1.161427 -1.634133 -2.517285 -1.120083
2013-01-03 -1.409665  0.516400  0.241388  0.064734
2013-01-04 -0.293723  0.722081  0.033310  0.797433
```

Existem dois métodos muito úteis de seleção que permitem localizar linhas (ou colunas) satisfazendo um critério de busca (`loc`) ou especificando suas posições (`iloc`). Com o `loc` são utilizados os rótulos das linhas e colunas. Abaixo, selecionam-se todas as linhas relativas as colunas `A` e `B`.

```
In [28]: df.loc[:,['A','B']]
Out[28]:
                   A         B
2013-01-01  1.795895  0.428990
2013-01-02  0.729177  1.448289
2013-01-03  0.276477 -1.167711
2013-01-04  0.088234  0.607176
2013-01-05 -1.780621  0.293291
2013-01-06 -0.499000 -2.246464
```

Analogamente, é possível selecionar um subconjunto de linhas e um subconjunto de colunas.

```
In [29]: df.loc['20130102':'20130104',['A','B']]
Out[29]:
                   A         B
2013-01-02  0.729177  1.448289
2013-01-03  0.276477 -1.167711
2013-01-04  0.088234  0.607176
```

Já com o `iloc`, os índices são usados para a seleção. Neste caso, são selecionadas as linhas 3 e 4 e as colunas 0 e 1.

```
In [30]: df.iloc[3:5,0:2]
Out[30]:
                   A         B
2013-01-04  0.088234  0.607176
2013-01-05 -1.780621  0.293291
```

O método `at` é utilizado para selecionar rapidamente uma célula do `DataFrame`. Por exemplo, abaixo é selecionado o elemento da primeira linha na coluna `A`.

```
In [32]: df.at[df.index[0], 'A']
Out[32]: 1.7958951465197248
```

Combinando DataFrames

A biblioteca *Pandas* possuí vários métodos para combinar `DataFrames` que são similares a operações em um banco de dados por meio de SQL. Um deles é o `merge`, que possibilita realizar a *junção* de `Dataframes`. Considere os `Dataframes` `df1` e `df2` construídos abaixo.

```
In [33]: df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'],
...                           'value': [1, 2, 3, 5]})
In [34]: df1
Out[34]:
  lkey  value
0  foo      1
1  bar      2
2  baz      3
3  foo      5
In [35]: df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'],
...                           'value': [5, 6, 7, 8]})
In [36]: df2
Out[36]:
  rkey  value
0  foo      5
1  bar      6
2  baz      7
3  foo      8
```

A junção de `df1` e `df2` nas colunas `lkey` e `rkey` pode ser feita da seguinte forma:

```
In [37]: df1.merge(df2, left_on='lkey', right_on='rkey')
Out[37]:
  lkey  value_x rkey  value_y
0  foo        1  foo        5
1  foo        1  foo        8
2  foo        5  foo        5
3  foo        5  foo        8
4  bar        2  bar        6
5  baz        3  baz        7
```

Pandas também possui a função `groupby`, a qual pode ser usada para uma ou mais das seguintes tarefas:

- Dividir os dados com base em algum critério.
- Aplicar a função para cada grupo.
- Combinar o resultado em um `DataFrame`.

O `groupby` suporta o agrupamento por uma lista de colunas, podendo ser aplicada uma função para cada agrupamento. Considere para um exemplo o seguinte `DataFrame`.

```
In [40]: df = pd.DataFrame({
   ...: 'A' : ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],
   ...: 'B' : ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
   ...: 'C' : np.random.randn(8),
   ...: 'D' : np.random.randn(8)})
In [41]: df
Out[41]:
     A      B         C         D
0  foo    one  0.699437 -1.241271
1  bar    one -0.478602 -1.282819
2  foo    two -0.798638 -0.814409
3  bar  three -0.934389  1.347837
4  foo    two -1.078957 -1.107251
5  bar    two -1.041093 -1.617251
6  foo    one -1.942070  0.910410
7  foo  three  0.095535 -0.223301
```

Então, o DataFrame é agrupado pelas colunas `A` e `B`, sendo aplicada a função `sum` em cada grupo resultante.

```
In [42]: df.groupby(['A', 'B']).sum()
Out[42]:
                  C         D
A   B
bar one   -0.478602 -1.282819
    three -0.934389  1.347837
    two   -1.041093 -1.617251
foo one   -1.242633 -0.330861
    three  0.095535 -0.223301
    two   -1.877595 -1.921660
```

Visualização de Dados em DataFrames

A biblioteca *Pandas* também possui uma série de facilidades para visualização de dados. A seguir, alguns exemplos são apresentados.

Primeiro, é necessário importar o módulo *pyplot*, da biblioteca *matplotlib*.

```
In [43]: import matplotlib.pyplot as plt
```

Abaixo, cria-se um *DataFrame* de exemplo com valores aleatórios.

```
In [44]: df = pd.DataFrame(np.random.randn(10, 4), index=
   ....: pd.date_range('1/1/2000', periods=10),
   ....: columns=['A', 'B', 'C', 'D'])
```

Então chama-se o método `plot`. A saída desse método é mostrada na Figura 6.2. Repare que de forma simples gerou-se um gráfico com vários elementos complexos, incluindo: várias linhas de cores diferentes, legenda (com os nomes das colunas do *DataFrame*) e eixo horizontal com rótulos temporais (provenientes da coluna de índices do *DataFrame*).

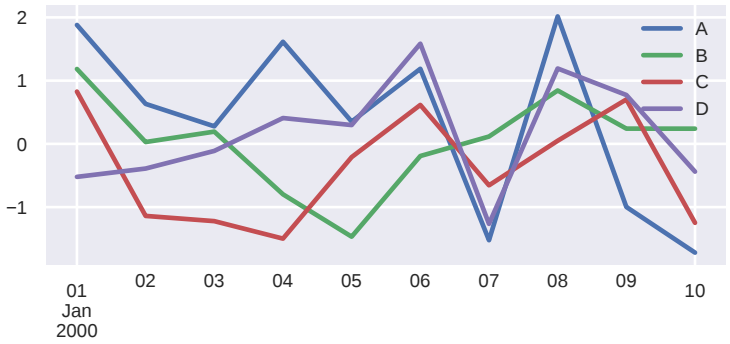


Figura 6.2: Exemplo de gráfico produzido com o método `plot`

.

```
In [45]: df.plot()
```

Por motivos de espaço, está fora do escopo deste texto discutir todos os aspectos da função plot. No entanto, salienta-se que vários outros tipos de gráficos podem ser criados. Entre eles: gráficos de barra, histogramas, *boxplots*, gráficos de dispersão e gráficos de pizza.

Nesta parte do minicurso, abordou-se *Pandas*, biblioteca desenvolvida na linguagem Python para facilitar o processo de leitura, manipulação e visualização de dados. A próxima parte relata algoritmos usados para reduzir a dimensionalidade desses dados, visando otimizar os processos de análise e extração de informação.

PARTE II - Redução de Dimensionalidade

Técnicas de aprendizado de máquina são utilizados para encontrar padrões ou tendências em dados. Em geral, esses padrões ou tendências são identificados por apresentar um conjunto de características semelhantes. Esse conjunto é representado computacionalmente através de um vetor de valores (inteiros, reais, lógicos, textos). O termo dimensionalidade é atribuído ao número de características de uma representação de padrões.

O objetivo desta parte é apresentar uma das tarefas mais recorrentes em análise de dados, a redução de dimensionalidade. Duas técnicas são discutidas e exemplificadas. Por fim, será mostrado como elas podem ser utilizadas para encontrar anomalias em matrizes de tráfego de redes.

Assume-se que os dados estão organizados em uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$. É comum, mas não obrigatório, interpretar as linhas de $\mathbf{X}$ como sendo *objetos* de interesse e as colunas representando *atributos*.

Da álgebra linear, sabe-se que o *rank* de $\mathbf{X}$ é o número de colunas (ou linhas) linearmente independentes. Por exemplo, se o *rank* de $\mathbf{X}$ for $r < \min\{n,d\}$, então é possível remover as colunas (ou linhas) que são formadas de combinações lineares das demais. A matriz resultante é menor, mas contém a mesma "informação" que $\mathbf{X}$.

Na prática, raramente encontra-se uma matriz de dados cujo *rank* seja menor que suas dimensões. No entanto, é comum encontrar matrizes com baixo *rank efetivo* [Roy and Vetterli, 2007]. Nesse caso, o *rank* real da matriz pode até ser máximo, mas tal

matriz pode ser "bem" aproximada por uma matriz cujo *rank r* seja muito menor que $\min\{n,d\}$. Para encontrar essa aproximação, técnicas de redução de dimensionalidade podem ser empregadas. Tais técnicas são interessantes, ou muitas vezes necessárias, por vários motivos. Entre eles:

1. Em muitas situações, os objetos de interesse são descritos por uma grande quantidade de atributos, quando na verdade um pequeno subconjunto desses atributos (ou uma combinação linear deles) seria suficiente para descrevê-los de maneira apropriada. Nesse caso, a redução de dimensionalidade auxilia na eficiência de armazenamento e no desempenho de alguns algoritmos.

2. Ao lidar com objetos em dimensionalidades muito altas, a intuição que muitas vezes é aplicada para lidar com objetos de uma, duas ou três dimensões não é mais válida. Esse fenômeno é conhecido como *a maldição da dimensionalidade* (sugere-se a leitura do Capítulo 6 de [Zaki and Jr, 2014]).

3. Há varias aplicações interessantes (dentro e fora da área de redes) baseadas na teoria *rank* efetivo de matrizes. Por exemplo, sistemas de recomendação e detecção de anomalias. Esse último será discutido mais adiante.

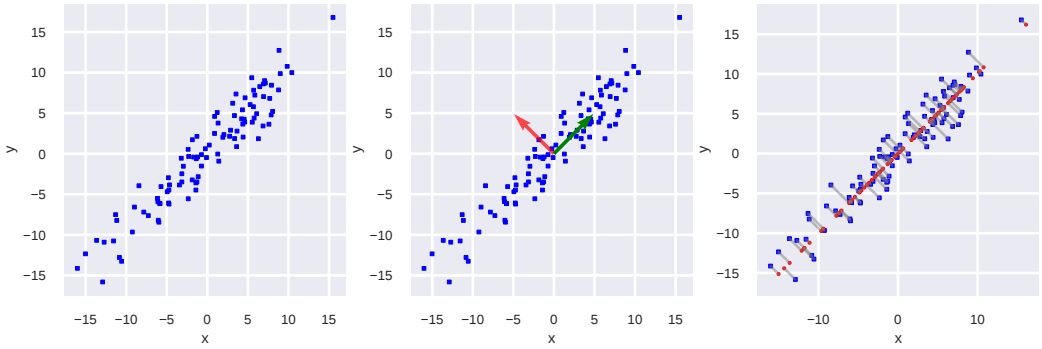
Duas técnicas populares, e de certa forma relacionadas, para redução de dimensionalidade são a análise de componentes principais e a decomposição em valores singulares, as quais são conhecidas como PCA (*Principal Component Analysis*) e SVD (*Singular Value Decomposition*), respectivamente.

A seguir, as técnicas PCA e SVD serão brevemente descritas, assim como utilizá-las de forma simples em Python. Em seguida, será apresentado como elas podem ser utilizadas para detectar anomalias em tráfego de uma rede, como proposto em [Lakhina et al., 2005]. Ao leitor interessado em mais detalhes e demonstrações matemáticas sobre os conceitos aqui abordados, sugere-se a leitura do Capítulo 7 de [Zaki and Jr, 2014].

PCA – Análise de Componentes Principais

A Figura 6.3a apresenta um conjunto de dados composto por uma coleção de pontos pertencentes ao $\mathbb{R}^2$. Pode-se perceber que os pontos estão basicamente distribuídos na vizinhança da reta $y = x$. A Figura 6.3b sobrepõe ao conjunto de pontos dois vetores ortogonais. Um deles, em verde, acompanha a direção em que a variância do conjunto de pontos é maior. Já a Figura 6.3c apresenta a projeção de cada ponto sobre a reta formada ao longo da direção de maior variância. Uma vez que a aproximação por uma reta parece razoável, pode-se pensar em substituir o conjunto de dados original pelas coordenadas de cada ponto no subespaço gerado pela reta. Dessa forma, a representação dos dados será simplificada, com uma pequena perda de informação. Para cada ponto, essa perda, ou erro, é representada pela linha cinza na Figura 6.3c, representando a distância entre o ponto original (em azul) e o ponto projetado no subespaço (em vermelho).

A técnica de Análise de Componentes Principais permite, de forma geral, executar os passos ilustrados pela Figura 6.3. Dada uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, representando n objetos do espaço d-dimensional (assumindo que $n \geq d$ e que o *rank* de $\mathbf{X}$ seja d), PCA permite:



(a) (b) (c)

Figura 6.3: Intuição da técnica de PCA: (a) conjunto de pontos no plano; (b) direções ortogonais de maiores variâncias; (c) projeções dos pontos no subespaço gerado na direção de maior variância.

1. encontrar d direções ortogonais que capturam a variância máxima do conjunto de dados, assim como na Figura 6.3b;

2. encontrar a variância dos dados associada a cada uma das direções ortogonais. A direção contendo a maior variância é denominada *primeira componente principal*. De forma geral, a direção associada à i-ésima maior variância é denominada i-ésima componente principal.

3. projetar os n objetos no subespaço r-dimensional ($r \leq d$) gerado pelas r primeiras componentes principais, assim como na Figura 6.3c. Além disso, é possível obter as coordenadas de cada objeto neste novo subespaço, permitindo substituir o conjunto de dados original por uma representação aproximada e mais compacta.

A biblioteca *Scikit-learn* do Python permite utilizar PCA de uma forma bem simples e conveniente.[5] A seguir, será ilustrado como aplicar PCA ao conjunto de dados da Figura 6.3. Primeiro, é necessário gerar o conjunto de dados, o que é feito no código abaixo utilizando a biblioteca *numpy*.[6] Em tal código, 100 observações de uma distribuição normal bivariada são geradas.

```
In [1]: import numpy as np
In [2]: np.random.seed(0)
In [3]: sigma = np.array([[41, 39], [39, 41]])
In [4]: x = np.random.multivariate_normal([0, 0], sigma, 100)
```

A seguir, é necessário importar a biblioteca e criar um objeto da classe `PCA`, passando como argumento r, i.e., o número de componentes principais que tem-se interesse.

```
In [5]: from sklearn.decomposition import PCA
In [6]: pca = PCA(n_components=1)
```

[5] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[6] http://www.numpy.org

Para aplicar PCA aos dados gerados, `x`, é necessário utilizar o método `fit`.

```
In [7]: pca.fit(x)
```

Agora, é possível ter acesso às *r* primeiras componentes principais e às variâncias associadas a cada uma delas, como indicado a seguir.

```
In [8]: pca.components_
Out[8]: array([[0.70436626, 0.70983672]])

In [9]: pca.explained_variance_
Out[9]: array([84.3970249])

In [10]: pca.explained_variance_ratio_
Out[10]: array([0.97590927])
```

No código acima, o atributo `components_` contém os vetores (direções) de cada componente principal. O atributo `explained_variance_` informa a variância associada a cada componente. Já o atributo `explained_variance_ratio_` apresenta a porcentagem da variância total dos dados que é explicada por cada componente. No caso do exemplo, a primeira componente captura cerca de 98% da variância total dos dados, e por isso, a aproximação dos dados por uma reta é razoável.

Por fim, para efetivamente reduzir dimensionalidade de `x`, pode-se utilizar o método `transform`, o qual faz a projeção dos pontos originais no subespaço gerado pelas *r* primeiras componentes principais. Esse recurso é exemplificado no código abaixo. Repare que a matriz original possui duas colunas, ao passo que a matriz de dados reduziada possui apenas uma.

```
In [12]: pca.transform(x)
Out[12]:
array([[-15.78520352],
       [ -8.7511212 ],
       [-16.71851443],
       [ -8.50796033],
       [  0.91610997],
       ...

In [13]: x
Out[13]:
array([[-11.55700386, -10.75668944],
       [ -8.43097572,  -3.94918933],
       [-10.83419594, -12.7887517 ],
       [ -5.85752955,  -6.16024396],
       [  0.24221484,   1.06341184],
       ...
```

SVD – Decomposição em Valores Singulares

Considere uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, e assuma que o *rank* de $\mathbf{X}$ seja l. A decomposição em valores singulares de $\mathbf{X}$ consiste em reescrever $\mathbf{X}$ como o produto de três matrizes da

seguinte maneira:

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T, \tag{1}$$

onde:

1. $\mathbf{U} \in \mathbb{R}^{n \times l}$ é uma matriz ortogonal. As colunas de $\mathbf{U}$ são denominadas de vetores singulares à esquerda.
2. $\mathbf{S} \in \mathbb{R}^{l \times l}$ é uma matriz diagonal, cujos valores da diagonal são números reais positivos e ordenados de maneira decrescente. Tais valores são denominados valores singulares.
3. $\mathbf{V} \in \mathbb{R}^{d \times l}$ é também uma matriz ortogonal. As colunas de $\mathbf{V}$ são denominadas vetores singulares à direita.

Por que esta decomposição é interessante? Dentre os vários motivos, dois destacam-se. Primeiro, pode-se demonstrar que é possível (e eficiente) fazer PCA utilizando SVD. Segundo, dado $r \leq l$, considere as matrizes:

1. $\mathbf{U}_r \in \mathbb{R}^{n \times r}$ sendo a matriz formada pelas primeiras r colunas de $\mathbf{U}$;
2. $\mathbf{S}_r \in \mathbb{R}^{r \times r}$ sendo a matriz quadrada formada pelos elementos das r primeiras linhas e r primeiras colunas de $\mathbf{S}$;
3. $\mathbf{V}_r \in \mathbb{R}^{d \times r}$ sendo a matriz formada pelas r primeiras colunas de $\mathbf{V}$; e
4. $\mathbf{X}_r = \mathbf{U}_r\mathbf{S}_r\mathbf{V}_r^T$.

É possível mostrar que $\mathbf{X}_r$ é a melhor aproximação, com *rank* r, de $\mathbf{X}$, ou seja, pode-se mostrar que $\mathbf{X}_r$ minimiza

$$f(x) = ||x - \mathbf{X}||_F, \tag{2}$$

no espaço de todas as matrizes com n linhas, d colunas e *rank* r. Na Equação (2), $||\cdot||_F$ denota a norma de *Frobenius*.

O poder da técnica SVD será ilustrado por meio de um exemplo de compressão de imagens. A Figura 6.4a apresenta a imagem, em escala de cinza, de um barco. A imagem tem 512 *pixels* de altura e largura. Uma vez que a imagem está em escala de cinza, esta pode ser interpretada como um matriz com 512 linhas e 512 colunas, onde cada célula da matriz indica a intensidade de cinza. Originalmente, são necessários 512×512 valores para representar a imagem.

Ao aplicar SVD e computar a matriz $\mathbf{X}_{40}$, a imagem da Figura 6.4b é obtida. Observe que a qualidade da imagem diminui, mas agora são necessário apenas $2 \times 40 \times 512 + 40$ valores para armazenar a imagem, i.e., 6.4 vezes menos em comparação com a imagem original.

Python permite aplicar SVD de uma maneira muito conveniente, por meio da biblioteca *numpy*.[7] No exemplo abaixo, uma matriz de números aleatórios é gerada. Após isso, os vetores e valores singulares são obtidos.

[7] `https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.svd.html`

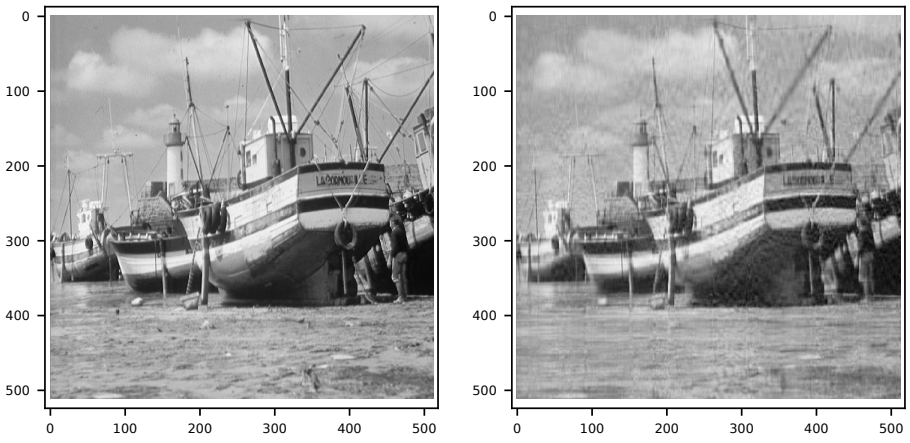


Figura 6.4: Utilização de SVD para compressão de imagens: (a) imagem original; (b) approximação da imagem utilizando uma matriz de *rank* 40.

```
In [1]: import numpy as np
In [2]: np.random.seed(0)
In [3]: X = np.random.random((4, 3))
In [4]: X
Out[5]:
array([[0.5488135 , 0.71518937, 0.60276338],
       [0.54488318, 0.4236548 , 0.64589411],
       [0.43758721, 0.891773  , 0.96366276],
       [0.38344152, 0.79172504, 0.52889492]])
In [6]: U, S, Vt = np.linalg.svd(X, full_matrices = False)

In [6]: U
Out[6]:
array([[-0.48665747,  0.08953811, -0.54332309],
       [-0.41275168,  0.81483266, -0.00407301],
       [-0.62010377, -0.2250203 ,  0.73183668],
       [-0.45636813, -0.52668447, -0.41133746]])

In [7]: S
Out[7]: array([2.21425506, 0.29626672, 0.21874633])

In [8]: Vt
Out[8]:
array([[-0.42376582, -0.64907895, -0.63175869],
       [ 0.65045995, -0.70346017,  0.2864361 ],
       [-0.63033672, -0.28955189,  0.72030224]])
```

É importante observar que no retorno do método da biblioteca *numpy* apenas os valores singulares são retornados, i.e., a diagonal da matriz **S**. Além disso, observe que o método retorna a matriz transposta de **V** e não a matriz **V** em si.

Exemplo: Minerando Anomalias de Tráfego

A redução da dimensionalidade foi aplicada por [Lakhina et al., 2005] para identificar anomalias de tráfego na rede. O exemplo apresentado a seguir é uma adaptação deste trabalho.

O problema de detecção de anomalias pode ser visto da seguinte forma: dada uma população P, tem-se o objetivo em dividir os elementos de P em um conjunto N de elementos "normais" e outro O, de indivíduos "anômalos". Uma maneira usual de atacar tal problema é utilizar aprendizado supervisionado para construir um modelo que separe N de O. O problema dessa abordagem é que ela requer dados rotulados para treinar o modelo, o que nem sempre está disponível. No entanto, se o fenômeno da baixa dimensionalidade puder ser observado (i.e., se os dados puderem ser organizado em uma matriz com baixo *rank* efetivo), é possível detectar anomalias de forma apropriada sem a necessidade de dados previamente rotulados.

Para este fim, assume-se que na população P, a maioria das observações seguem o comportamento "normal". Então, pode-se usar o fenômeno de baixa dimensionalidade para separar as anomalias das não anomalias. Para tal, é necessário obter um modelo (linear) de baixa dimensão para os dados. Assim, todo comportamento que não puder ser bem descrito pelo modelo, provavelmente foge da normalidade e portanto, pode ser enquadrado como uma anomalia. Na prática, dada uma matriz de dados $\mathbf{X}$, os seguintes passos devem ser seguidos:

1. Obtenha a decomposição em valores singulares de $\mathbf{X}$

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T; \tag{3}$$

2. Obtenha uma aproximação de $\mathbf{X}$ com baixo *rank* efetivo, r

$$\mathbf{X}_r = \mathbf{U}_r\mathbf{S}_r\mathbf{V}_r^T; \tag{4}$$

3. Compute a porção dos dados que não pode ser explicada por $\mathbf{X}_r$

$$\mathbf{O} = \mathbf{X} - \mathbf{X}_r; \tag{5}$$

4. Identifique as linhas (ou colunas) de $\mathbf{O}$ com as maiores normas: essas linhas (colunas) correspondem às anomalias.

Para aplicar essa metodologia à detecção de anomalias no tráfego de uma rede de computadores, primeiro, é necessário modelar tal tráfego como uma matriz. Usualmente, isso é feito da seguinte forma: o tráfego entre cada par origem-destino (OD) é medido em intervalos fixos de tempo. Após isso, uma matriz $\mathbf{X}$ é construída, onde o elemento (i, j) da matriz é referente ao tráfego do par OD j no intervalo de tempo i. Como caso de estudo, considere o tráfego observado na rede *Abilene*[8] durante uma semana de setembro de 2003. A rede contém 120 pares origem-destino, para os quais o tráfego da origem para

[8] `https://uit.stanford.edu/service/network/internet2/abilene`

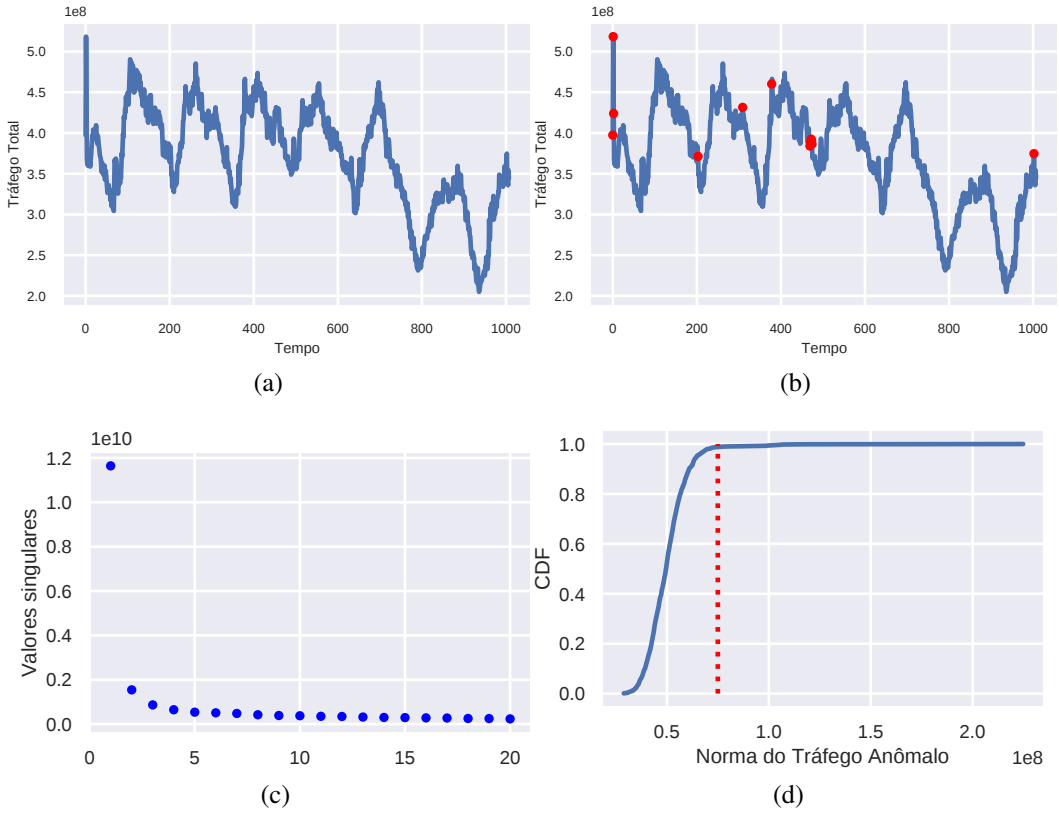


Figura 6.5: (a) Tráfego total na rede; (b) Tráfego total na rede, onde os pontos em vermelho são considerados anômalos; (c) Norma ℓ_2 das linhas da matriz de $\mathbf{O}$; (d) Distribuição acumulada (CDF – *Cumulative Distribution Function*) das normas ℓ_2 das linhas de $\mathbf{O}$.

o destino foi medido em intervalos de 10 minutos. A Figura 6.5a apresenta o tráfego total medido em cada intervalo, i.e., a soma dos elementos de cada linha da matriz de tráfego. A pergunta que deseja-se responder é: há algum ponto de tempo em que é possível afirmar que o tráfego na rede é anômalo? Responder essa pergunta é uma tarefa desafiadora, uma vez que os dados contêm padrões de variação diários e semanais. Além disso, uma anomalia pode surgir por diferentes motivos, e.g., um ataque à rede, falha de equipamento ou uma demanda repentina devido a um evento externo.

De acordo com a metodologia sumarizada acima, é preciso encontrar uma representação de baixa dimensionalidade para $\mathbf{X}$. Mas como definir r? Uma heurística usual consiste em analisar o gráfico dos valores singulares de $\mathbf{X}$ e definir r como sendo a abscissa do "joelho" da curva, i.e., o ponto em que a curva para de decrescer de forma brusca. Para o conjunto de dados em questão, o gráfico é apresentado na Figura 6.5c, o qual indica que $r = 6$ é uma escolha razoável e conservadora.

A Figura 6.5d apresenta a distribuição das normas ℓ_2 das linhas de $\mathbf{O} = \mathbf{X} - \mathbf{X}_6$. Na figura, pode-se perceber que valores acima de 0.75×10^8 são infrequentes. Por isso, escolheu-se esse valor como limiar para indicar se um dado intervalo de tempo apresenta comportamento anômalo ou não. Há na matriz $\mathbf{O}$ 13 linhas (cada uma referente a um intervalo de tempo) cujas normas excedem esse limiar. Os intervalos de tempo associ-

ados a cada uma dessas linhas são destacadas junto ao tráfego total da rede na Figura 6.5b. Pode-se perceber que, em alguns dos casos, o tráfego anômalo é referente a uma observação globalmente discrepante. No entanto, este fato não é uma regra. Há também observações discrepantes locais e outras que são consideradas anômalas por não poderem ser explicadas por um modelo linear de baixa dimensão.

Nesta parte do minicurso foram apresentadas PCA (*Principal Component Analysis*) e SVD (*Singular Value Decomposition*), duas formas de reduzir a dimensionalidade dos vetores de características usados para representar padrões nos dados. Um exemplo onde essa técnica foi aplicada na área de redes também foi relatado. Na parte seguinte, serão apresentados algoritmos que conseguem identificar automaticamente padrões nos dados.

PARTE III - Aprendizado Não Supervisionado

Aprendizado não supervisionado é uma área de aprendizado de máquina que tem por objetivo *aprender* a partir de dados não rotulados. A tarefa mais comum em aprendizado não supervisionado, e que será o foco desse texto, é a de agrupamento (ou *clustering*).

Nesse contexto, o objeto de estudo tipicamente é uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, onde n é o número de objetos a serem agrupados e d é a dimensionalidade de cada objeto. Informalmente, o problema de interesse pode ser descrito como dividir as n linhas de $\mathbf{X}$ em k grupos naturais, ou seja, de forma que:

- elementos do mesmo grupo sejam "similares"; e
- elementos de grupos diferentes não sejam "similares".

Há na literatura uma variedade de algoritmos para resolver o problema de agrupamento, com base em heurísticas ou de forma exata. Cada algoritmo possui características específicas com relação à eficiência (tamanho do problema que consegue-se resolver em um tempo razoável), aos tipos de grupos naturais que conseguem identificar e às diferentes noções de "similaridade" entre objetos.

Considere como exemplo a Figura 6.6. Um ser humano é capaz de identificar diferentes grupos naturais em cada uma das subfiguras sem grandes dificuldades. Apesar do exemplo da figura parecer trivial, um olhar mais cuidadoso revela vários desafios para abordagens automatizadas. Entre eles: grupos de tamanhos não uniformes, grupos não convexos e grupos com densidades diferentes. Ao longo desse texto, estes exemplos serão utilizados para mostrar que poucos algoritmos conseguem aprender "bem" em todas essas situações.

Dessa forma, pretende-se dar uma visão geral sobre o problema de agrupamento por meio da apresentação das características de alguns dos métodos populares, a saber: K-Means; Agrupamento aglomerativo; Agrupamento baseado em densidade; e Agrupamento espectral. Além disso, serão mostrados recursos da biblioteca *Scikit-learn*, do Python, para a utilização de tais algoritmos.

[9] `https://scikit-learn.org/stable/modules/clustering.html`

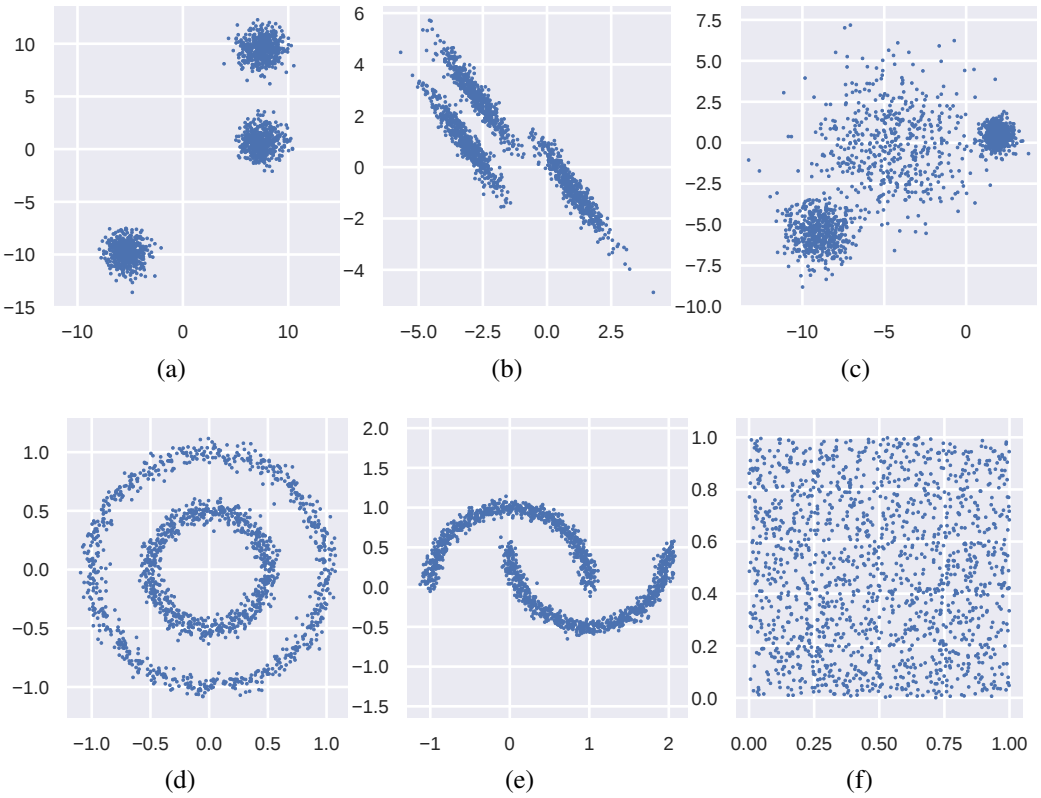


Figura 6.6: Exemplos de vários tipos de grupos naturais: (a) grupos circulares; (b) grupos alongados e de mesmo tamanho; (c) grupos com densidades diferentes; (d) grupos circulares e concêntricos; (e) grupos não convexos; (f) sem estrutura de grupos. Exemplos baseados na documentação da biblioteca *Scikit-learn*.[9]

Por fim, o contexto de redes de computadores será abordado por meio de um exemplo, onde algoritmos de agrupamento serão utilizados para detectar "comunidades" em topologias de rede.

K-means

O algoritmo K-means é uma das abordagens mais populares para tarefa de agrupamento de dados. Atualmente, o termo K-means é utilizado para uma série de heurísticas baseadas no trabalho de [Forgy, 1965], as quais consistem em variações e melhoramentos do algoritmo proposto em 1965. Formalmente, dados $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ ($\mathbf{x}_i \in \mathbb{R}^d$) e um inteiro $k \geq 2$, o K-means tem o objetivo de minimizar

$$f(\mathbf{c}_1, \ldots, \mathbf{c}_k) = \sum_{i=1}^{n} \min_{1 \leq j \leq k} ||\mathbf{x}_i - \mathbf{c}_j||^2, \tag{6}$$

onde $||\cdot||$ representa a norma ℓ_2 e $\mathbf{c}_j \in \mathbb{R}^d$ ($1 \leq j \leq k$). Intuitivamente, a otimização dessa função objetivo consiste em encontrar k pontos no espaço d-dimensional, denominados centroides, que "representem" bem o conjunto de pontos. Na Figura 6.6a, por exemplo, para $k = 3$, uma boa solução é aquela que coloca cada centroide no centro de um aglomerado.

O problema de minimização descrito acima é NP-difícil. No entanto, as heurísticas conhecidas na literatura funcionam razoavelmente bem para conjuntos de dados não muito grandes. A biblioteca *Scikit-learn*, do Python, traz por padrão uma das variações mais populares, o *K-means++*. Um exemplo, referente a geração e aplicação do método para os dados da Figura 6.6a, é apresentado a seguir.

Primeiro, é preciso importar os módulos necessários e gerar os pontos.

```
In [1]: from sklearn import datasets, cluster
In [2]: import numpy as np
In [3]: np.random.seed(0)
In [4]: X, _ = datasets.make_blobs(n_samples=1500, random_state=8)
```

Segundo, constrói-se um objeto da classe `KMeans`. No caso do exemplo, é informado que deseja-se agrupar os pontos em 3 grupos. Então, o método `fit` é utilizado para executar o algoritmo como os parâmetros especificados no construtor.

```
In [5]: kmeans = cluster.KMeans(n_clusters = 3)
In [6]: kmeans.fit(X)
```

Após a execução, é possível identificar o grupo de cada ponto do conjunto de dados e o valor da função objetivo (soma dos quadrados das distâncias de cada ponto para o respectivo centroide) por meio dos atributos `labels_` e `inertia_`.

```
In [7]: kmeans.labels_
Out[7]: array([2, 2, 2, ..., 2, 1, 1], dtype=int32)
In [8]: kmeans.inertia_
Out[8]: 2959.7867290413565
```

Repare que a implementação da biblioteca *Scikit-learn* exige o que o número de grupos seja fornecido. No caso do exemplo, a escolha de $k = 3$ é imediata, mas o que fazer quando os dados não podem ser facilmente visualizados ou quando não há uma intuição sobre o número de grupos naturais?

Uma técnica popular para tal tarefa consiste em observar o valor da função objetivo para diferentes valores de k. Aumentar k implica em diminuir o valor da função; no entanto, ao se incrementar o valor de k, se o decremento da função objetivo não for significativo, pode-se entender que não há necessidade de realizar o incremento. A Figura 6.7 apresenta a aplicação dessa técnica para o conjunto de pontos da Figura 6.6a.

Por fim, a Figura 6.8 apresenta o resultado da aplicação do K-means para os conjuntos de pontos da Figura 6.6. A técnica não é indicada para situações em que os grupos naturais possuem geometria não convexa, densidade variável ou grupos alongados (e.g., Figuras 6.8b, 6.8c, 6.8d e 6.8e).

Agrupamento Aglomerativo

O agrupamento aglomerativo usa uma abordagem *bottom-up*, criando uma sequência de partições aninhadas, que pode ser vista como uma hierarquia de grupos, também chamada de dendrograma. A Figura 6.9 mostra um exemplo de dendrograma.

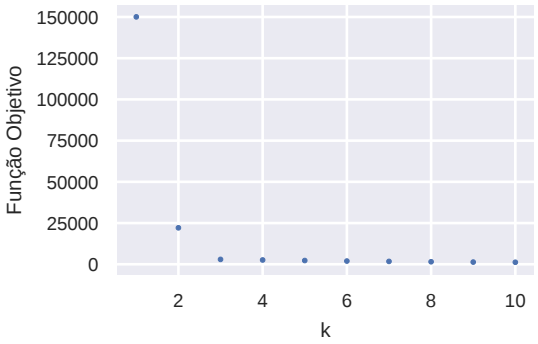


Figura 6.7: Valor da função objetivo para diferentes valores de k. Pode-se observar que o decremento do valor da função para $k > 3$ é quase nulo. Esse resultado sugere que $k = 3$ é uma escolha razoável para o número de grupos para o conjunto de dados em questão.

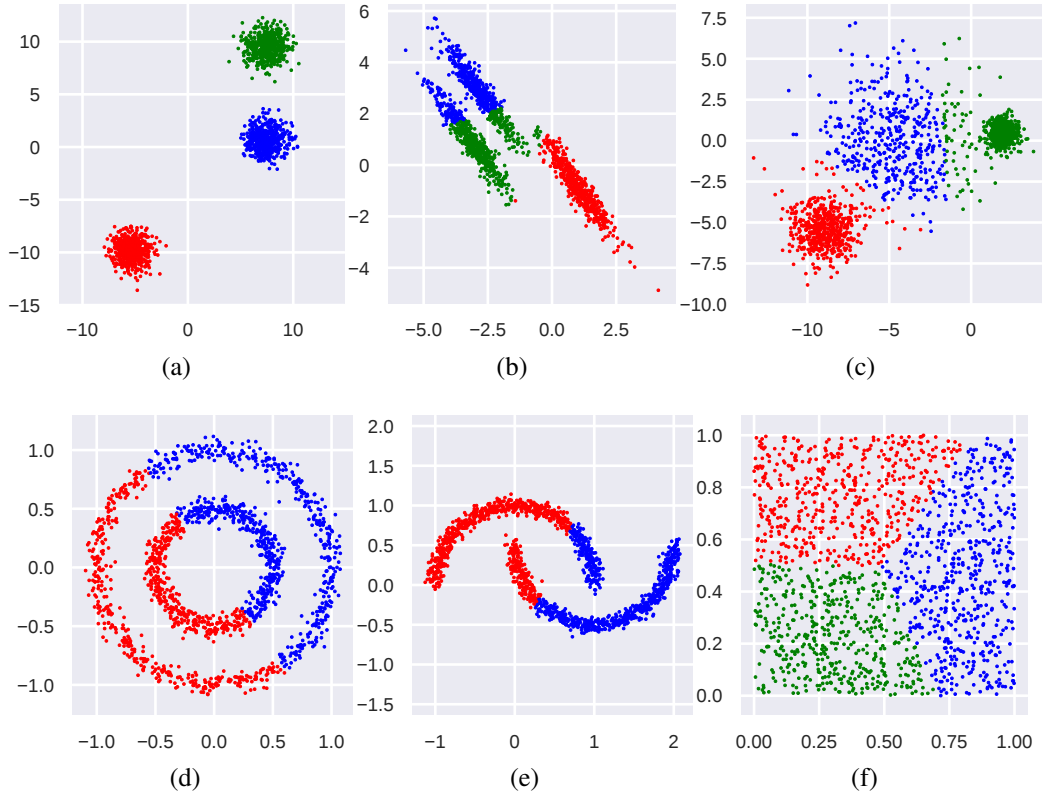


Figura 6.8: Aplicação do algoritmo K-means para os exemplos da Figura 6.6. Cores diferentes representam grupos diferentes identificados pelo algoritmo. Valores de k, em ordem, são: 3, 3, 3, 2, 2 e 3.

Dada uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, cujas linhas são denotadas por $\mathbf{x}_1, \ldots, \mathbf{x}_n$, e um inteiro k, o agrupamento aglomerativo funciona da seguinte forma:

1. Na inicialização, cada $\mathbf{x}_i$ se torna um grupo $\{\mathbf{x}_i\}$.
2. O próximo passo consiste de um laço onde os dois grupos mais similares são aglomerados em um só até que restem apenas k grupos.

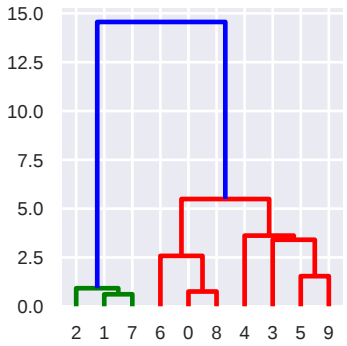


Figura 6.9: Exemplo de um dendrograma

Note que, dependendo de como é calculada a distância entre os grupos, o resultado do agrupamento pode ser diferente. Algumas variantes de como esse cálculo de distância pode ser feito são listadas a seguir. Em todos os casos, $\delta(\mathbf{x},\mathbf{y})$ denota a distância Euclidiana entre os pontos $\mathbf{x}$ e $\mathbf{y}$.

- *Single Link*: A distância entre dois grupos C_1 e C_2 é dada pela distância mínima entre um ponto em C_1 e outro em C_2

$$\delta(C_1,C_2) = min\{\delta(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in C_1, \mathbf{y} \in C_2\}.$$

- *Complete Link*: A distância entre dois grupos C_1 e C_2 é dada pela distância máxima entre um ponto em C_1 e outro em C_2

$$\delta(C_1,C_2) = max\{\delta(\mathbf{x},\mathbf{y}) \mid \mathbf{x} \in C_1, \mathbf{y} \in C_2\}.$$

- *Group Average*: A distância entre dois grupos C_1 e C_2 é dada pela distância média par a par entre pontos em C_1 e C_2

$$\delta(C_1,C_2) = \frac{\sum_{\mathbf{x}\in C_1}\sum_{\mathbf{y}\in C_2}\delta(\mathbf{x},\mathbf{y})}{n_1 n_2},$$

  onde n_1 e n_2 são o número de pontos em C_1 e C_2, respectivamente.

- *Ward's Method*: A distância entre dois grupos C_1 e C_2 é dada pela aumento na soma dos erros quadrados, de cada ponto para o centroide do grupo, quando dois grupos são unidos

$$\delta(C_1,C_2) = \left(\frac{n_1 n_2}{n_1+n_2}\right)\|\mu_1-\mu_2\|^2,$$

  onde $\mu_1 = \frac{1}{n_1}\sum_{\mathbf{x}\in C_1}\mathbf{x}$ e $\mu_2 = \frac{1}{n_2}\sum_{\mathbf{x}\in C_2}\mathbf{x}$

A biblioteca *Scikit-learn*, do Python, traz a implementação do Agrupamento Aglomerativo com todas essas variações do cálculo da distância entre grupos. Um exemplo, referente a geração e aplicação do método para os dados da Figura 6.6a, é apresentado a seguir.

Primeiro, é preciso importar os módulos necessários e gerar os pontos.

```
In [1]: from sklearn import datasets, cluster
In [2]: import numpy as np
In [3]: np.random.seed(0)
In [4]: X, _ = datasets.make_blobs(n_samples=1500, random_state=8)
```

Segundo, constrói-se um objeto da classe `AgglomerativeClustering`. No caso do exemplo, é informado que deseja-se agrupar os pontos em 3 grupos com o cálculo da distância entre os grupos sendo feito com o método de *Ward*. Então, o método `fit` é utilizado para executar o algoritmo com os parâmetros especificados no construtor.

```
In [5]: agg = cluster.AgglomerativeClustering(n_clusters = 3,
   ...:        linkage = 'ward')
In [6]: agg.fit(X)
```

Após a execução, é possível identificar o grupo de cada ponto do conjunto de dados por meio do atributo `labels_`.

```
In [7]: agg.labels_
Out[7]: array([2, 2, 2, ..., 2, 1, 1], dtype=int32)
```

Por fim, a Figura 6.10 apresenta o resultado da aplicação do Agrupamento Aglomerativo para os conjuntos de pontos da Figura 6.6.

Agrupamento Baseado em Densidade

O Agrupamento Baseado em Densidade mais popular, chamado DBSCAN, tem o objetivo de encontrar grupos de alta densidade (pontos por região) que sejam isolados uns dos outros.

Considere uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$, cujas linhas são denotadas por $\mathbf{x}_1, \ldots, \mathbf{x}_n$, um número real ε e um inteiro η. Em alto nível, o DBSCAN funciona da seguinte forma:

1. Para cada ponto $\mathbf{x}$, encontre todos os vizinhos numa distância máxima ε. Os pontos que tiverem mais de η vizinhos dentro desta distância são denominados *core points*;
2. Encontre conjuntos maximais de *core points* tais que cada *core point* esteja em uma ε-vizinhança de ao menos um outro *core point*;
3. Associe a cada ponto que não é um *core points* ao grupo mais próximo, se a distância for inferior a ε;
4. Todos os outros pontos são rotulados como ruído.

A vantagem do Agrupamento Baseado em Densidade é a facilidade de encontrar grupos não convexos. Mas sua aplicação é um pouco mais complicada, já que não há definição do número de grupos. O resultado é baseado nos valores de ε e η, os quais devem ser fornecidos pelo usuário.

A biblioteca *Scikit-learn*, do Python, traz a implementação do DBSCAN. Um exemplo, referente a geração e aplicação do método para os dados da Figura 6.6a, é apresentado a seguir.

Primeiro, é preciso importar os módulos necessários e gerar os pontos.

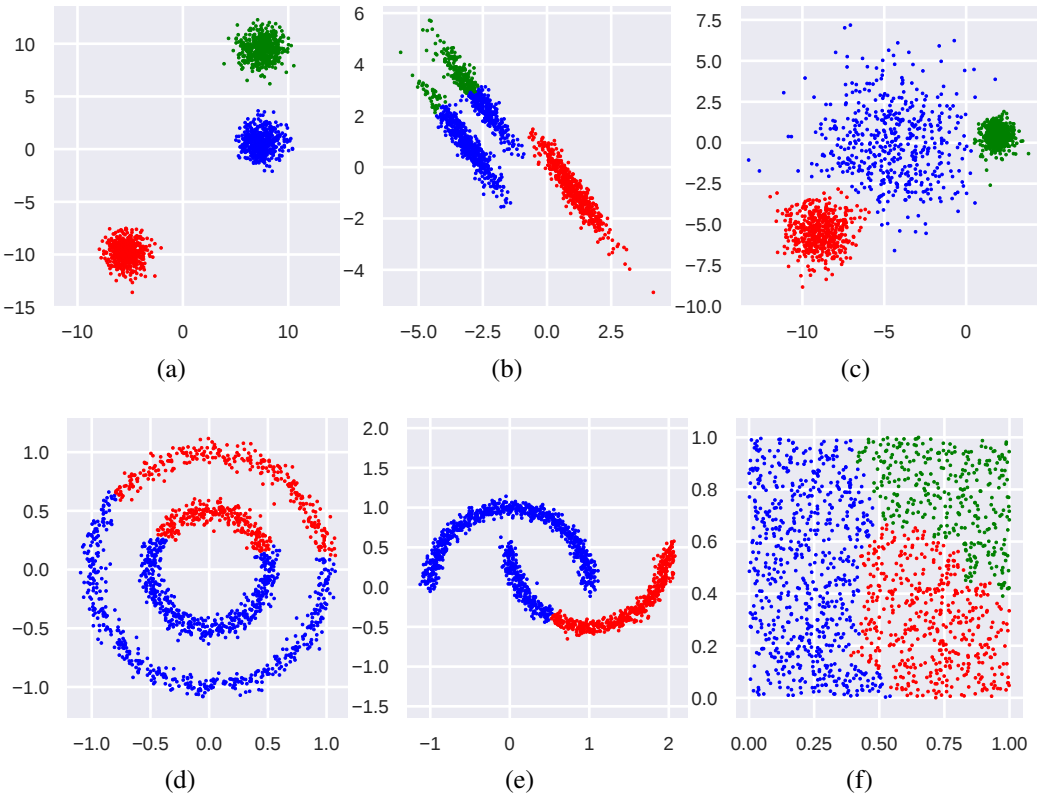


Figura 6.10: Aplicação do Agrupamento Aglomerativo usando o método de *Ward* para os exemplos da Figura 6.6. Cores diferentes representam grupos diferentes identificados pelo algoritmo. Valores de k, em ordem, são: 3, 3, 3, 2, 2 e 3.

```
In [1]: from sklearn import datasets, cluster
In [2]: import numpy as np
In [3]: np.random.seed(0)
In [4]: X, _ = datasets.make_blobs(n_samples=1500, random_state=8)
```

Segundo, constrói-se um objeto da classe `DBSCAN`. No caso do exemplo, é informado que deseja-se agrupar os pontos utilizando $\varepsilon = 1.5$ e $\eta = 5$. Então, o método `fit` é utilizado para executar o algoritmo como os parâmetros especificados no construtor.

```
In [5]: dbs = cluster.DBSCAN(eps = 1.5, min_samples = 5)
In [6]: dbs.fit(X)
```

Então, é possível identificar o grupo de cada ponto do conjunto de dados e o índice dos *core-points* por meio dos atributos `labels_` e `core_sample_indices_`.

```
In [7]: dbs.labels_
Out[7]: array([0, 0, 0, ..., 0, 2, 2], dtype=int32)
In [8]: dbs.core_sample_indices_
Out[8]: array([   0,    1,    2, ..., 1497, 1498, 1499], dtype=int32)
```

A Figura 6.11 apresenta o resultado da aplicação do DBSCAN para os conjuntos de pontos da Figura 6.6.

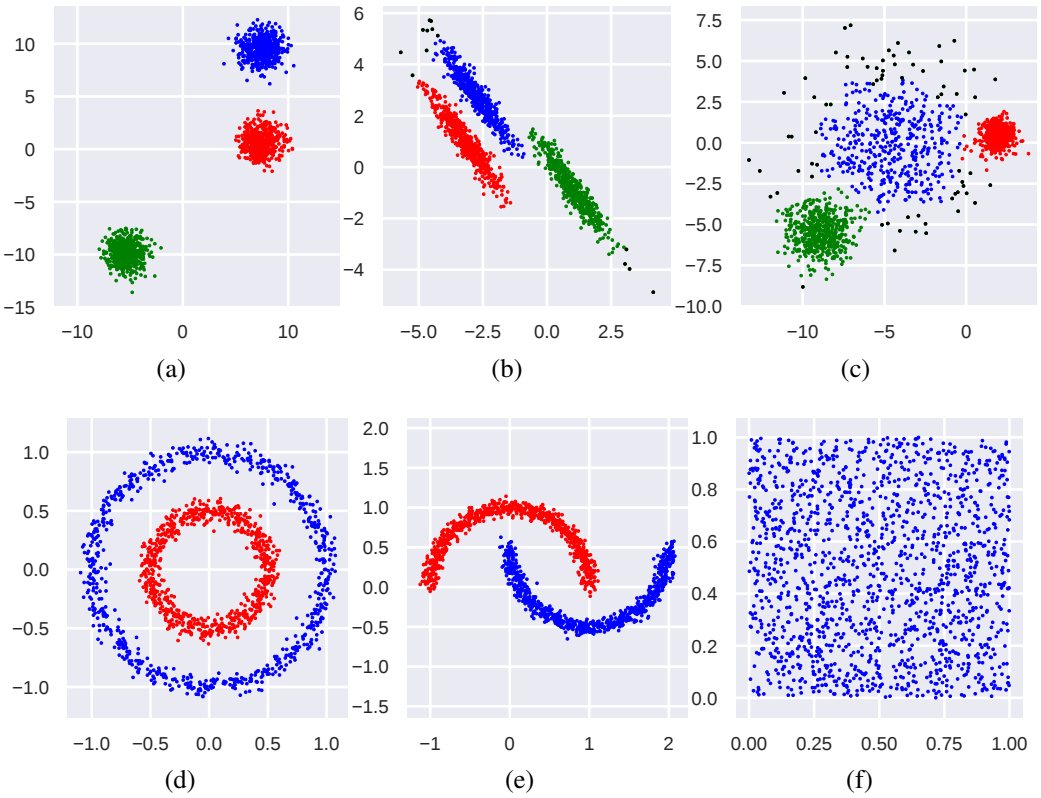


Figura 6.11: Aplicação do DBSCAN para os exemplos da Figura 6.6. Cores diferentes representam grupos diferentes identificados pelo algoritmo. Pontos em preto são considerados ruídos pelo algoritmo. Valores de ε, em ordem, são: 1.50, 0.35, 0.80, 0.23, 0.20, 0.30. Valores de η, em ordem, são: 5, 5, 10, 50, 50, 5.

Uma limitação do DBSCAN é a sensibilidade aos parâmetros que devem ser fornecidos, especialmente ε, cuja uma pequena variação pode mudar drasticamente a quantidade e forma dos grupos. Não há um procedimento único e bem definido para a escolha desses parâmetros. Na prática, recorre-se a experimentação, tentativa e erro.

Agrupamento Espectral

O método de agrupamento espectral foi projetado para detectar grupos naturais em grafos. Apesar de ter sua fundamentação teórica voltada para tais objetos, é possível utilizá-lo para dados em uma organização matricial, assim como para os métodos descritos anteriormente. Inicialmente, será apresentada uma intuição do porquê do nome *espectral*, e então será discutido como a técnica pode ser utilizada quando o objeto de estudo não é um grafo. Por fim, será demonstrado como pode-se utilizar a técnica por meio da biblioteca *Scikit-learn* do Python.

Considere o exemplo da Figura 6.12a. Pode-se perceber que, apesar de ser conexo, o grafo em questão possui três "grupos" distintos: $\{0,1,2,3,4\}$, $\{5,6,7,8\}$ e $\{9,10,11,12\}$. Cada nó do grafo está mais conectado com outros nós dentro do que fora do grupo a que pertence. O objetivo do agrupamento espectral é justamente capturar essa intuição, ou seja, particionar um grafo em subgrafos (ou comunidades) que sejam "bem" conectados.

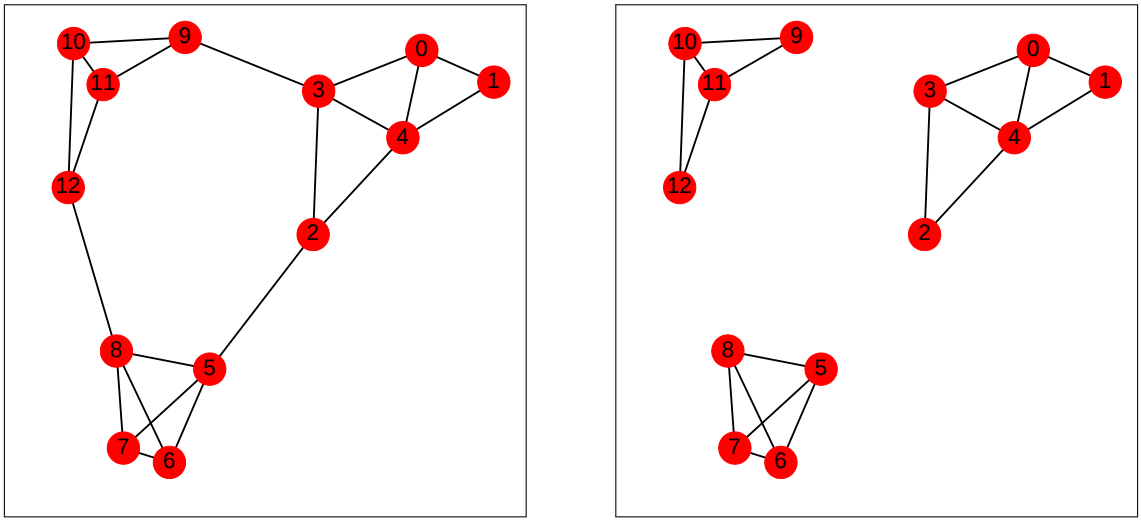


Figura 6.12: (a) grafo conexo com estrutura em comunidades. (b) grafo não conexo.

Para motivar como a técnica funciona, considere o grafo da Figura 6.12b. O grafo consiste do mesmo exemplo da Figura 6.12a, apenas não contendo as arestas que conectam os diferentes grupos. Seja $\mathbf{A}$ a matriz de adjacências e $\mathbf{D}$ a matriz de graus do grafo em questão. A matriz Laplaciana do grafo é definida como sendo $\mathbf{L} = \mathbf{D} - \mathbf{A}$. No caso do exemplo, tem-se que que a matriz $\mathbf{L}$ é dada por:

$$\begin{pmatrix}
3 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & -1 & -1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 3 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 3 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 2
\end{pmatrix}$$

Pode-se perceber que tal matriz tem uma estrutura bloco diagonal. Além disso, tem-se que os três seguintes vetores

$$\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T,$$
$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T \text{ e}$$
$$\mathbf{z} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

são autovetores de $\mathbf{L}$ associados ao autovalor 0.

De forma geral, se a matriz $\mathbf{L}$ possuir k autovetores (linearmente independentes) associados ao autovalor 0, então o grafo refente a tal matriz possui k componentes conexas. Esse fato em si não é útil na prática, pois raramente os grafos de interesse possuem a estrutura de comunidades formada por subgrafos desconexos (como na Figura 6.12b). No entanto, o exemplo acima apresenta uma intuição de que o espectro da matriz Laplaciana está fortemente relacionado com a estrutura de comunidades do grafo (se tal estrutura existir).

Os detalhes de como a técnica funciona estão além do escopo deste texto. O leitor interessado pode consultar o texto de [Zaki and Jr, 2014]. É importante ressaltar que a matriz Laplaciana (e sua versão normalizada) também pode ser construída para grafos cujas arestas têm peso. Esse fato permite que a técnica de agrupamento espectral seja aplicada a bases de dados arbitrárias, bastando para isso considerar que a matriz de afinidades entre os objetos do conjunto de dados seja interpretada como a matriz de adjacências de um grafo. Essa abordagem é seguida pela biblioteca *Scikit-learn*[10] cuja documentação também possui mais detalhes sobre a parte teórica do algoritmo. A aplicação do agrupamento espectral pela biblioteca *Scikit-learn* é muito similar aos casos anteriores. O código para importar as bibliotecas necessárias e gerar um conjunto de dados de exemplo é mostrado abaixo.

```
In [1]: from sklearn import datasets, cluster
In [2]: import numpy as np
In [3]: np.random.seed(0)
In [4]: X, _ = datasets.make_blobs(n_samples=1500, random_state=8)
```

Para construir um modelo de agrupamento espectral, fornecendo o número de grupos desejados, e para aplicar utilizar o modelo em um conjunto de dados, tem-se:

```
In [5]: esp = cluster.SpectralClustering(n_clusters=3)
In [6]: esp.fit(X)
```

Por fim, os rótulos referentes às linhas da matriz $\mathbf{X}$ podem ser acessados da mesma forma que nos exemplos anteriores, por meio do atributo `labels_`.

```
In [7]: esp.labels_
Out[7]: array([1, 1, 1, ..., 1, 0, 0], dtype=int32)
```

[10] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html

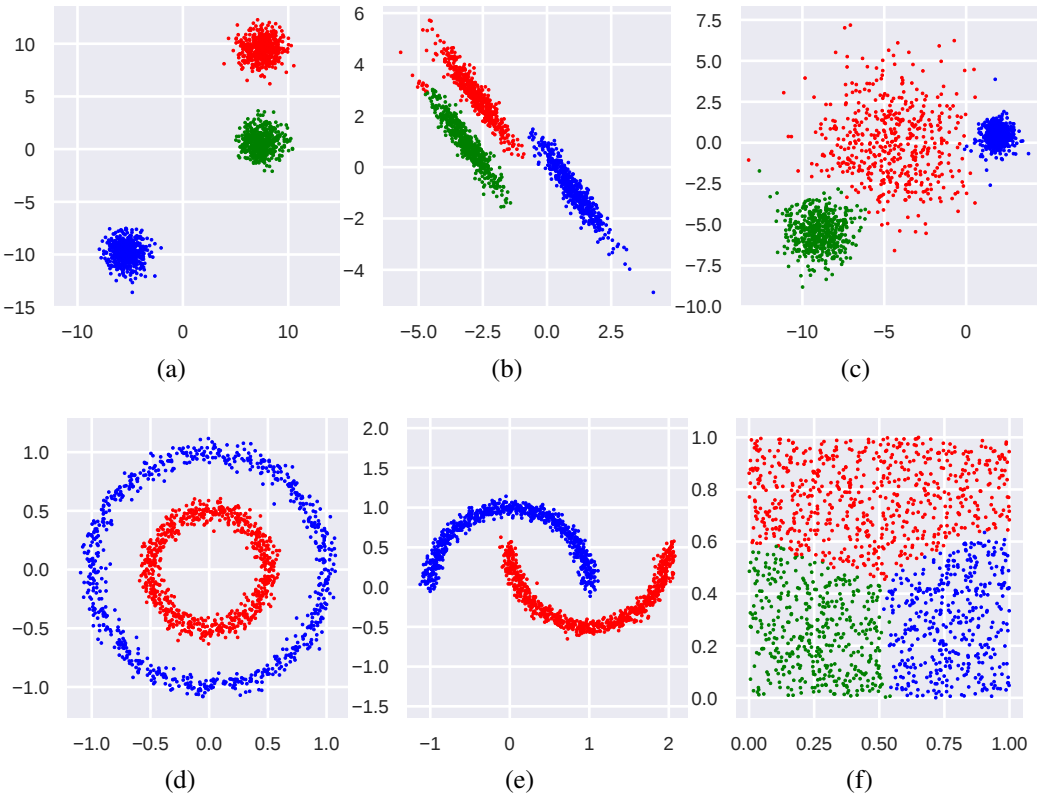


Figura 6.13: Aplicação do agrupamento espectral para os exemplos da Figura 6.6. Cores diferentes representam grupos diferentes identificados pelo algoritmo. Valores de k, em ordem, são: 3, 3, 3, 2, 2 e 3. Além disso, o parâmetro `affinity='nearest_neighbors'` foi utilizado.

O resultado da aplicação do agrupamento espectral nos exemplos da Figura 6.6 é apresentado na Figura 6.13. Pode-se perceber que entre todas as técnicas exemplificadas, esta foi a que proporcionou melhores resultados.

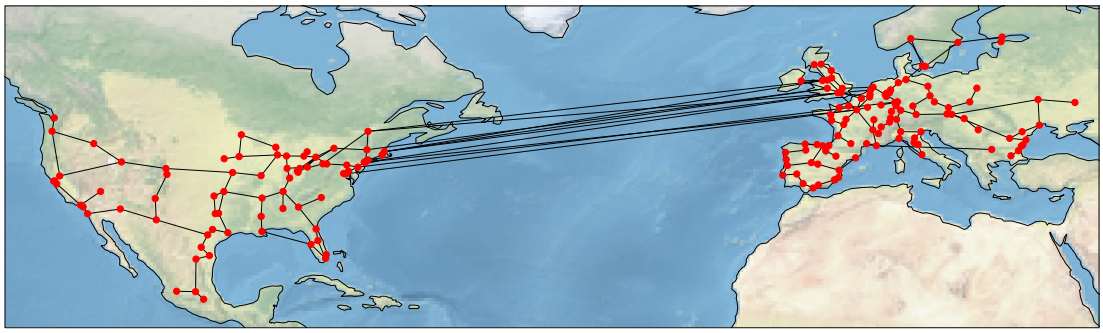
De forma geral, o agrupamento espectral é capaz de encontrar grupos significativos e próximos dos grupos naturais dos dados. Por outro lado, a técnica tem um custo computacional maior e também possui mais dificuldades para determinar o valor correto de k, i.e., o número de grupos desejado.

Exemplo: Detectando Comunidades em Redes

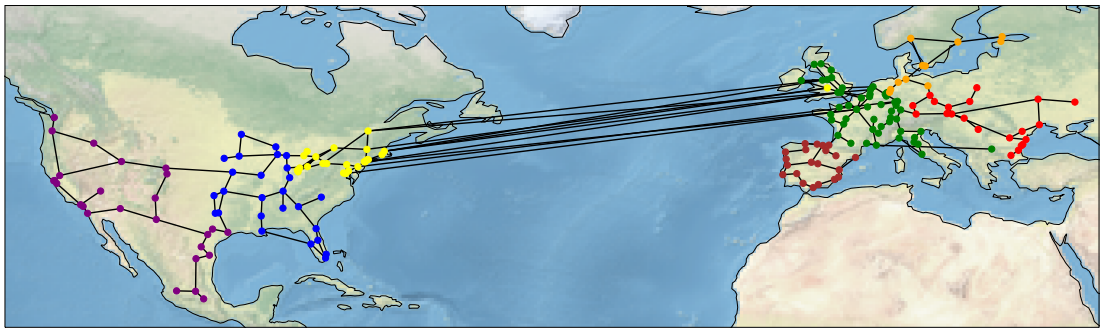
Em várias situações, dados provenientes de redes de computadores podem ser representados por meio de grafos. Exemplos são: topologias de redes (em diferentes camadas) e redes sociais. Neste contexto, uma tarefa comum de interesse consiste em encontrar partes da rede cujos elementos dentro de cada parte sejam "bem" conectados entre si, mas que sejam "fracamente" conectados com elementos de partes diferentes; em outras palavras, tem-se interesse em detectar comunidades na rede.

De forma geral, o problema de detecção de comunidades em grafos é desafiador, uma vez que até mesmo o conceito de "comunidade" é muito subjetivo. A seguir, tem-se

o objetivo de mostrar, por meio de um exemplo, que algumas das técnicas de agrupamento apresentadas nas seções anteriores podem ser utilizadas para o problema de detecção de comunidades em grafos, com resultados razoáveis.



(a)



(b)

Figura 6.14: (a) Grafo representando a rede da empresa Cogent em agosto de 2010. Cada nó do grafo representa um PoP da rede e cada aresta representa uma conexão entre PoPs (ao nível de IP). (b) Resultado da aplicação do agrupamento espectral no grafo anterior para $k = 6$. Cada cor representa um grupo diferente identificado pelo algoritmo.

A Figura 6.14a apresenta um exemplo de grafo baseado nos dados obtidos do projeto *The Internet Topology Zoo* [Knight et al., 2011]. Cada nó do grafo representa um ponto de presença (PoP) da rede da empresa Cogent (referente ao mês de Agosto de 2010) e cada aresta representa uma conexão entre PoPs (ao nível de IP). Além das informações topológicas, os dados contêm a geolocalização de cada PoP, o que permite estimar o custo de comunicação entre dois PoPs por meio da distância entre eles.

Como discutido anteriormente, o método de agrupamento espectral é fundamentado em teoria de grafos, podendo ser utilizado para encontrar comunidades em grafos de forma natural. A Figura 6.14b apresenta o resultado do agrupamento do grafo da Figura 6.14a utilizando $k = 6$ (i.e., seis grupos). Neste caso, a biblioteca *Scikit-learn* foi utilizada considerando a estrutura do grafo em si e também a distância física entre os nós do grafo. Pode-se perceber, por meio de uma inspeção visual, que os grupos identificados pelo algoritmo são correlacionados com regiões do mapa em comunidades distintas do grafo.

Esta parte explicou métodos que identificam padrões em dados não rotulados, foram reportados os algoritmos K-means, agrupamento aglomerativo, agrupamento baseado

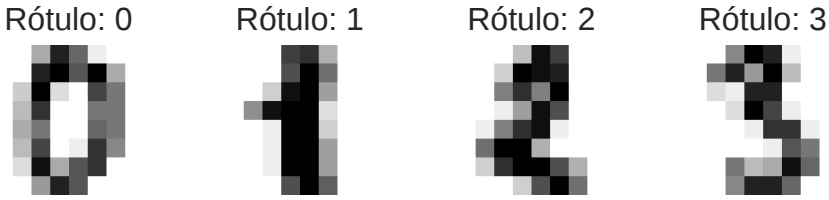


Figura 6.15: Exemplo de problema de classificação. Como associar cada imagem, referente a um dígito escrito à mão, ao número que ela representa? Nesse caso, cada imagem é representada por uma matriz 8×8, cujos valores definem uma escala de cinza. Essas imagens podem então ser representadas por vetores de 64 dimensões. Esse problema possui um total de 10 classes, uma vez que há 10 dígitos distintos. Exemplo baseado na documentação da biblioteca *Scikit-learn*.[11]

em densidade e agrupamento espectral, bem como um exemplo da aplicação deste último método para identificar diferentes regiões no globo terrestre, a partir das informações georreferenciadas dos PoPs e links de uma empresa de telecomunicações. Na sequência, serão apresentados métodos que aprendem a partir de dados previamente rotulados.

PARTE IV - Aprendizado Supervisionado

Aprendizado supervisionado é uma área de aprendizado de máquina que tem o objetivo de *aprender* (ou treinar) a partir de dados (vetor de características) rotulados, para então classificar novos dados em função do conhecimento adquirido no treinamento. Há duas tarefas principais: regressão e classificação. Nesse contexto, o objeto de estudo pode ser representado por meio de uma matriz $\mathbf{X} \in \mathbb{R}^{n \times d}$ e por um vetor $\mathbf{y}$, onde:

- n é o número de objetos;
- d é a dimensionalidade de cada objeto; e
- y_i é o rótulo (ou valor) associado a $\mathbf{x}_i$ (i-ésima linha de $\mathbf{X}$).

Um exemplo do problema de classificação é ilustrado na Figura 6.15, o qual consiste da tarefa de associar imagens, referentes a dígitos escritos à mão, aos números que representam.

Dados $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, a tarefa de classificação, foco deste texto, consiste em encontrar um modelo f que aprenda, a partir dos vetores de características, os rótulos. No entanto, há requisitos importantes com relação ao modelo desejado; os dois principais são:

1. se $(\mathbf{x}, y) \in D$, deseja-se que $f(\mathbf{x}) = y$; e
2. a função aprendida deve ser *generalizável*. Se um novo $\mathbf{x}$ for observado (i.e, um $\mathbf{x}$ que não esteja em D), deseja-se que $f(\mathbf{x})$ seja o rótulo real de $\mathbf{x}$.

[11] `https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html`

Satisfazer apenas o requisito 1 é fácil. Satisfazer 1 e 2 simultaneamente, por outro lado, é uma tarefa difícil. Para ver que apenas o requisito 1 é fácil de ser satisfeito, considere o exemplo a seguir. Dados $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ e um vetor não rotulado que deseja-se classificar, $\mathbf{w}$, defina o seguinte classificador:

1. se $\mathbf{w}$ estiver em D, retorne a classe associada;

2. se $\mathbf{w}$ não estiver em D, retorne um rótulo aleatório (entre os valores válidos).

Observe que o requisito 1 é totalmente satisfeito, mas f generaliza de forma pobre, pois a chance de se acertar o rótulo real de $\mathbf{w}$ (quando este não estiver em D) é $\frac{1}{m}$, onde m é o número de classes. Esse exemplo também ilustra um dos maiores problemas em aprendizado de máquina: *overfitting*. Tal fenômeno ocorre quando o modelo explica tão bem o conjunto de dados usado durante o aprendizado, que é incapaz de generalizar para dados não vistos.

No restante do texto, são apresentados alguns dos classificadores mais populares e como utilizá-los por meio da biblioteca *Scikit-learn* do Python. Além disso, são mostradas técnicas de como evitar o problema de *overfitting* e uma aplicação no contexto de classificação de matrizes de tráfego, a qual é baseada no trabalho de [Trois et al., 2018].

Naive Bayes

A ideia do classificador Naive Bayes é criar um modelo probabilístico para encontrar a classe de um dado $\mathbf{x} \in \mathbb{R}^d$. Mais especificamente, tem-se interesse em calcular $P(C_k \mid \mathbf{x})$, onde C_k é a k-ésima possível classe ($k = 1, \ldots, K$) relativa ao problema de classificação. A quantidade $P(C_k \mid \mathbf{x})$ define a probabilidade de C_k ser a classe associada a $\mathbf{x}$. Assim, o modelo probabilístico é utilizado para determinar a classe de $\mathbf{x}$ como sendo aquela que maximize tal probabilidade.

Da teoria de probabilidade, e de uma maneira informal, sabe-se que a probabilidade condicional da classe C_k dado um vetor $\mathbf{X}$ é dada por

$$P(C_k \mid \mathbf{x}) = \frac{P(C_k, x_1, \ldots, x_d)}{P(\mathbf{x})}. \tag{7}$$

Na prática, tem-se interesse apenas no numerador dessa equação, uma vez que o denominador não depende de C_k. Daí,

$$\begin{aligned} P(C_k, x_1, \ldots, x_d) &= P(x_1, \ldots, x_d, C_k) \\ &= P(x_1 \mid x_2, \ldots, x_d, C_k) P(x_2, \ldots, x_d, C_k) \\ &= P(x_1 \mid x_2, \ldots, x_d, C_k) P(x_2 \mid x_3, \ldots, x_d, C_k) P(x_3, \ldots, x_d, C_k) \\ &= \ldots \\ &= P(x_1 \mid x_2, \ldots, x_d, C_k) P(x_2 \mid x_3, \ldots, x_d, C_k) \ldots P(x_{d-1} \mid x_d, C_k) P(x_d \mid C_k) P(C_k) \end{aligned}$$

A hipótese ingênua do classificador, e daí o nome *Naive Bayes*, consiste de assumir que $P(x_i \mid x_{i+1},\ldots,x_d,C_k) = P(x_i \mid C_k)$. Dessa forma:

$$P(C_k \mid x_1,\ldots,x_d) \propto P(C_k,x_1,\ldots,x_d) = P(C_k)\prod_{i=1}^{d} P(x_i \mid C_k).$$

Para estimar o valor de $P(C_k \mid \mathbf{x})$, faz-se necessário saber apenas como calcular $P(x_i \mid C_k)$. Há várias formas de realizar tal cálculo. Geralmente, assume-se que os dados são provenientes de alguma distribuição de probabilidade (e.g., Normal, Multinomial e Bernoulli), e então, técnicas de inferência estatística são utilizadas para estimar os parâmetros de tal modelo. Após isso, o modelo pode ser utilizado para obter a classe de qualquer $\mathbf{x}$. O trabalho de [Metsis et al., 2006] apresenta detalhes de como esse procedimento pode ser realizado para diferentes distribuições de probabilidade, assim como uma aplicação do método para detecção de *spam* em *e-mails*.

Por fim, para estimar a classe de $\mathbf{x}$, denotada por $\hat{y}$, tem-se:

$$\hat{y} = \underset{z\in\{C_1,\ldots,C_K\}}{\operatorname{argmax}} \; P(z)\prod_{i=1}^{d} P(x_i \mid z)$$

A biblioteca *Scikit-learn* permite a utilização do classificador Naive Bayes de uma forma bem simples. O exemplo a seguir, mostra como treinar um modelo e utilizá-lo para predizer a classe de objetos. O conjunto de dados utilizado no exemplo é o mesmo da Figura 6.15.

Primeiro, é preciso importar as bibliotecas necessárias.

```
In [1]: from sklearn import datasets
In [2]: from sklearn.naive_bayes import GaussianNB
```

Abaixo, o conjunto de dados referente ao conjunto de dados usado na Figura 6.15 é carregado.

```
In [3]: X, y = datasets.load_digits(return_X_y = True)
In [4]: X
Out[4]:
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
In [5]: y
Out[5]: array([0, 1, 2, ..., 8, 9, 8])
```

Então, pode-se construir o classificador. Neste caso, utilizando uma versão que faz uso da distribuição Normal.

```
In [6]: model = GaussianNB()
In [7]: model.fit(X, y)
```

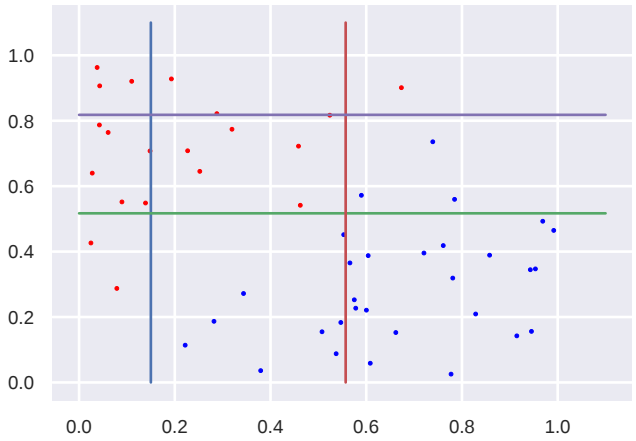


Figura 6.16: Aplicação de uma árvore de decisão em pontos no plano. Os pontos em azul são da classe 1, e os em vermelho da classe -1. As retas são as divisões feitas pela Árvore de Decisão treinada com esses pontos, mostrada na Figura 6.17.

Uma vez que o modelo é treinado, pode-se utilizar o método `predict` para predizer a classe de um conjunto de dados.

```
In [8]: model.predict(X)
Out[8]: array([0, 1, 8, ..., 8, 9, 8])
```

No exemplo acima, repare que a qualidade do classificador não foi avaliada. Além disso, apenas como título de exemplo, o método `predict` foi aplicado aos mesmos dados utilizados para a construção do modelo. Mais adiante, técnicas para avaliação e teste de modelos são apresentadas.

Árvores de Decisão

A ideia geral de uma Árvores de Decisão é criar uma árvore onde os nós internos são divisões no espaço de dados e os nós folha são rotulados com a classe mais frequente na região representada por esse nó. Como mostrado na Figura 6.16, o algoritmo de construção de uma Árvore de Decisão faz as divisões mais "puras" possíveis no espaço, de tal forma que os pontos estejam bem separados de acordo com suas respectivas classes. A Figura 6.17 mostra a Árvore de Decisão treinada com os pontos da Figura 6.16. Cada nó da Árvore de Decisão é um teste em algum atributo, e cada aresta é a saída daquele teste (verdadeiro ou falso). Note que, para classificar um novo ponto, basta caminhar na Árvore de Decisão verificando os condicionais até chegar em um nó folha, onde é dada a classe do ponto.

A forma mais utilizada para particionar o espaço de forma pura utiliza o GINI. Esse índice mede o quanto os dados são heterogêneos. O GINI é dado por: $1 - \sum_{i=1}^{d} p_i^2$, onde p_i é a frequência relativa de cada classe em cada nó. Quanto mais o GINI se aproxima de zero, mais o nó é considerado puro. Se o GINI se aproxima de 1, o nó é considerado impuro.

A biblioteca *Scikit-learn* permite a utilização do classificador Árvore de Decisão de uma forma bem simples. O exemplo a seguir, mostra como treinar um modelo e utilizá-

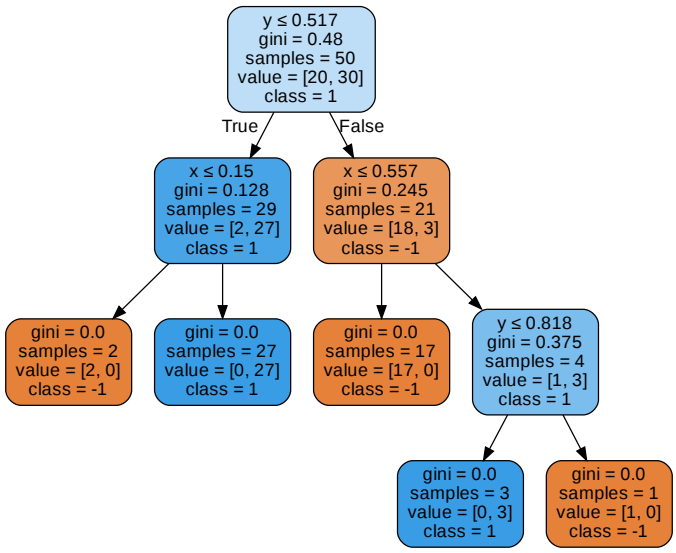


Figura 6.17: Árvore de Decisão construída com os pontos da Figura 6.16

lo para predizer a classe de objetos. O conjunto de dados utilizado no exemplo é o mesmo da Figura 6.15.

Primeiro, é preciso importar as bibliotecas necessárias.

```
In [1]: from sklearn import datasets
In [2]: from sklearn.tree import DecisionTreeClassifier
```

Abaixo, o conjunto de dados referente ao conjunto de dados usado na Figura 6.15 é carregado.

```
In [3]: X, y = datasets.load_digits(return_X_y = True)
In [4]: X
Out[4]:
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
In [5]: y
Out[5]: array([0, 1, 2, ..., 8, 9, 8])
```

Então, pode-se construir o classificador e treinar o modelo com base nos dados.

```
In [6]: model = DecisionTreeClassifier()
In [7]: model.fit(X, y)
```

Uma vez que o modelo é treinado, pode-se utilizar o método `predict` para predizer a classe de um conjunto de dados.

```
In [8]: model.predict(X)
Out[8]: array([0, 1, 2, ..., 8, 9, 8])
```

Florestas Aleatórias

Uma técnica bastante utilizada em classificação é a combinação de classificadores. É possível mostrar que, a combinação de n classificadores independentes utilizando o voto de maioria (a classe escolhida por pelo menos $\frac{n}{2}+1$ classificadores) melhora a acurácia da classificação. Na prática, é difícil ter uma quantidade grande de classificadores independentes, por isso são utilizados classificadores com baixa correlação.

A técnica de Florestas Aleatórias faz uso dessa filosofia, combinando uma sequência de Árvores de Decisão com baixa correlação. Para conseguir essa baixa correlação, aleatoriedade é introduzida no processo de criação das árvores, tanto na seleção dos objetos que serão utilizados para treinamento, como na seleção de atributos a serem utilizados como candidatos a critério de divisão em cada nó interno da árvore.

A biblioteca *Scikit-learn* permite a utilização do classificador Florestas Aleatórias. O exemplo a seguir mostra como treinar um modelo e utilizá-lo para predizer a classe de objetos. O conjunto de dados utilizado no exemplo é o mesmo da Figura 6.15.

Primeiro, é preciso importar as bibliotecas necessárias.

```
In [1]: from sklearn import datasets
In [2]: from sklearn.ensemble import RandomForestClassifier
```

Abaixo, o conjunto de dados é carregado.

```
In [3]: X, y = datasets.load_digits(return_X_y = True)
```

Então, pode-se construir o classificador; neste caso, 100 árvores de decisão serão utilizadas.

```
In [6]: model = RandomForestClassifier(n_estimators = 100)
In [7]: model.fit(X, y)
```

Uma vez que o modelo é treinado, pode-se utilizar o método `predict` para predizer a classe de um conjunto de dados.

```
In [8]: model.predict(X)
Out[8]: array([0, 1, 2, ..., 8, 9, 8])
```

SVM – Support Vector Machine

O SVM (*Support Vector Machine*) é um dos classificadores mais populares da literatura. Tal popularidade se deve, principalmente, à boa qualidade dos resultados proporcionados pelo classificador em diversos tipos de problemas de classificação, até mesmo em problemas difíceis e não lineares.

Há duas versões principais do SVM, uma que gera um modelo linear, i.e., que divide o conjunto de pontos por meio um plano (ou hiperplano), e uma que gera um

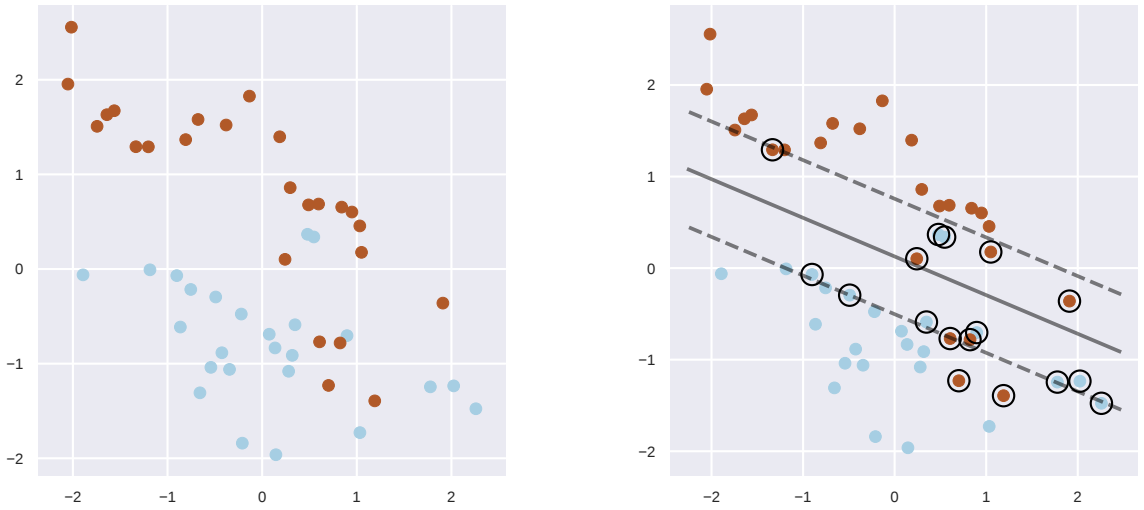


Figura 6.18: (a) conjunto de pontos a serem separados (i.e., classificados); (b) aplicação do classificador SVM ao conjunto de pontos. A linha sólida representa o plano de separação, as linhas tracejadas representam as margens do classificador e os pontos circulados são os vetores de suporte.

modelo não linear, a qual é necessária para problemas mais complexos. Aqui, apenas a versão linear será discutida, uma vez que a versão não linear é mais complexa e está fora do escopo do texto.

A Figura 6.18a mostra um conjunto de pontos pertencentes a duas classes distintas (indicadas pelas cores) a serem separados. Observe que os pontos podem ser separados por uma reta de forma quase perfeita. De fato, o objetivo do SVM é encontrar uma reta (ou um hiperplano no caso geral) que "melhor" separe o conjunto de pontos de acordo com as duas classes.

Mas o que significa "melhor" separar? No caso de um problema que pode perfeitamente ser separado por um modelo linear, o melhor modelo, ou hiperplano, é aquele que tem a maior margem, ou seja, que fica o mais distante possível dos pontos pertencentes as duas classes. Quando o problema não é linearmente separável, o melhor hiperplano é aquele que possui a maior margem, mesmo que a separação não seja perfeita, e que também minimize penalidades advindas de violações das margens. A ideia principal é que uma margem muito pequena implica na existência de vários modelos, dos quais, muitos são passíveis de generalização pobre. Por outro lado, uma margem muito grande pode penalizar bons modelos de uma forma desnecessária. Os conceitos de margem, hiperplano de separação e vetores de suporte são ilustrados na Figura 6.18b.

A dinâmica mencionada no parágrafo anterior é capturada pelo problema de otimização que o SVM pretende resolver. Dado um conjunto de treinamento $D = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$, com cada $\mathbf{x}_i \in \mathbb{R}^d$ $y_i \in \{-1, 1\}$, o objetivo é encontrar o hiperplano de separação $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$, cujos parâmetros são obtidos resolvendo o seguinte problema de otimização:

$$\min_{\mathbf{w},b,\zeta} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\zeta_i,$$

com restrições

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \zeta_i \text{ e } \zeta_i \geq 0,\ i = 1,\ldots,n.$$

No problema acima, C é uma constante e deve ser fornecida pelo usuário. Tal constante influencia diretamente no tamanho da margem do classificador, e portanto, na qualidade do modelo resultante da solução do problema de otimização. Um valor grande de C tende a diminuir o tamanho da margem, ao passo que um valor pequeno de C tende a aumentar a margem. Qual o valor de C adequado? A resposta desta pergunta está diretamente relacionado ao fenômeno de *overfitting* e será abordada na próxima seção.

Na biblioteca *Scikit-learn*, no mesmo molde dos classificadores apresentados anteriormente, o SVM pode ser utilizado de forma conveniente. De início, é necessário importar as bibliotecas, como mostrado abaixo.

```
In [1]: from sklearn import datasets
In [2]: from sklearn.svm import SVC
```

Após isso, deve-se criar o modelo, como os devidos parâmetros e ajustar tal modelo aos dados. No caso abaixo, cria-se um modelo *linear* com constante $C = 1$. A escolha deste C foi feita de forma arbitrária. Mais adiante, será apresentado como o valor correto de C deve ser selecionado.

```
In [3]: X, y = datasets.load_digits(return_X_y = True)
In [4]: model = SVC(kernel = 'linear', C = 1)
In [5]: model.fit(X, y)
```

Uma vez que o modelo é treinado, a função `predict` pode ser usada para predizer a classe de outros objetos.

```
In [6]: model.predict(X)
Out[6]: array([0, 1, 2, ..., 8, 9, 8])
```

Repare que no exemplo acima, o SVM foi utilizado para um problema de classificação não binária (há 10 classes nesse problema). Essa generalização não foi abordada neste texto. O leitor interessado pode consultar [Zaki and Jr, 2014].

Avaliação, Treino e Teste

No contexto de classificação, três perguntas importantes, e relacionadas, são:

1. Como avaliar um classificador?
2. Como obter os hiperparâmetros e um classificador (e.g., C do SVM ou o número de estimadores em uma floresta aleatória)?

3. Como evitar o problema de *overfitting*?

Quantificar o desempenho de um classificador é uma tarefa importante em. Além de possibilitar afirmar se um modelo é razoável ou não, tal tarefa é importante para a comparação de modelos diferentes. Neste contexto, considere $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ como sendo um conjunto no qual se pretende avaliar um classificador, denotado por μ, em um problema de classificação binária (i.e., $y_i \in \{0,1\}$).[12] De acordo com a classe real de cada $\mathbf{x}_i$, i.e., y_i, e a classe estimada de $\mathbf{x}_i$, i.e., $\mu(\mathbf{x}_i)$, definem-se as quatro seguintes quantidades:

- $TP = |\{(\mathbf{x}_i, y_i) \mid \mu(\mathbf{x}_i) = 1 \text{ e } y_i = 1\}|$;
- $FP = |\{(\mathbf{x}_i, y_i) \mid \mu(\mathbf{x}_i) = 1 \text{ e } y_i = 0\}|$;
- $FN = |\{(\mathbf{x}_i, y_i) \mid \mu(\mathbf{x}_i) = 0 \text{ e } y_i = 1\}|$;
- $TN = |\{(\mathbf{x}_i, y_i) \mid \mu(\mathbf{x}_i) = 0 \text{ e } y_i = 0\}|$.

Com essas quatro quantidades, é possível construir a *Matriz de Confusão* do classificador, assim como mostrado abaixo.

Predição \ Classe Real	Positiva	Negativa
Positiva	TP	FP
Negativa	FN	TN

Tabela 6.1: Estrutura de uma Matriz de Confusão. As colunas se referem as classes e as linhas as predições.

Tomando com base a Matriz de Confusão e as definições anteriores, há quatro métricas principais para avaliar ou comparar o desempenho de classificadores:

1. *Accuracy*:
$$\frac{TP+TN}{TP+TN+FP+FN},$$
i.e., do total de amostras, a fração que teve a classe corretamente predita;

2. *Precision*:
$$\frac{TP}{TP+FP},$$
i.e., dos que são preditos como pertencentes à classe positiva, a fração que realmente é pertencente à classe positiva;

3. *Recall*:
$$\frac{TP}{TP+FN},$$
i.e., dos que são pertencentes à classe positiva, a fração que é predita como pertencente à classe positiva;

[12] Assume-se que 1 é a classe positiva e 0 é a classe negativa.

4. *F-score*:
$$2\frac{pr}{p+r},$$
que é a média harmônica entre *Precision* (p) e *Recall* (r).

Como mencionado anteriormente, um bom classificador é aquele capaz de generalizar, ou seja, é esperado que tal classificador tenha um bom desempenho (de acordo com as métricas descritas) quando for utilizado em observações que não estiverem presentes no conjunto de treinamento. Mas dado um conjunto de treinamento $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, como garantir que um classificador treinado em D seja generalizável? Uma estratégia simples e bastante utilizada é "fingir" que uma parte de D não é conhecida! Em outras palavras, seleciona-se aleatoriamente uma fração dos pares em D (comumente por volta de 20%); denote essa porção D'. Após isso, treine o classificador escolhido em $D \setminus D'$. Use o conjunto não utilizado para treinamento (i.e., D') para testes. Se o classificador não tiver *overfitting*, então, ele deverá ter desempenho similar em D' e $D \setminus D'$.

Uma pergunta que ainda deve ser respondida é referente aos hiperparâmetros de um classificador. Por exemplo, como definir o C para o SVM ou o número de estimadores para uma floresta aleatória? Comumente, para evitar *overfitting*, faz-se uso de validação cruzada. Seja D o conjunto de treinamento (suponha que conjunto de testes já foi devidamente separado) e considere Θ como sendo um vetor de valores candidatos para os hiperparâmetros de um classificador escolhido. Então, proceda da seguinte maneira:

1. Particione, aleatoriamente, D em k partes: $D_1, \ldots, D_k$
2. Para cada $\theta \in \Theta$
   (a) Para cada $i \in \{1, \ldots, k\}$
      i. Treine o classificador em $D \setminus D_i$ e θ.
      ii. Avalie o classificador em D_i.
   (b) Calcule a média do desempenho dos k classificadores obtidos
3. Escolha o θ que proporcione os melhores resultados médios.

Repare que no procedimento acima, o particionamento de treino e teste é repetido novamente dentro do conjunto de treino. O objetivo desta etapa é, novamente, evitar *overfitting*.

A biblioteca *Scikit-learn* permite que todos os passos acima sejam de forma simples e com poucas linhas de código. A seguir, um exemplo mostrando um conjunto mínimo de etapas que devem ser seguidas para treinar um classificador de forma apropriada. O exemplo consiste de utilizar o SVM em um conjunto de dados de exemplos da própria *Scikit-learn*. Inicialmente, deve-se importar as bibliotecas necessárias, como mostrado abaixo.

```
In [1]: import numpy as np
In [2]: from sklearn.svm import SVC
In [3]: from sklearn.model_selection import GridSearchCV,
```

```
   ...:                                        train_test_split
In [4]: from sklearn import datasets
In [5]: from sklearn.preprocessing import MinMaxScaler
In [6]: from sklearn.metrics import confusion_matrix, accuracy_score,
   ...:                               precision_score, recall_score,
   ...:                               f1_score
```

O conjunto de dados de exemplo contém 569 observações, 30 atributos e duas classes. O método `train_test_split` permite dividir o conjunto de dados em porções distintas para treino e teste. Neste caso, 20% dos dados foram selecionados para teste.

```
In [7]: X, y = datasets.load_breast_cancer(return_X_y=True)
In [8]: XTrain, XTest, yTrain, yTest = train_test_split(X, y,
   ...:                                                 test_size = 0.2)
```

Uma prática comum em classificação consiste em normalizar a matriz de dados. Isso garante que todas as colunas da matriz de dados estejam na mesma ordem de grandeza. Há várias formas de conduzir esse processo. Abaixo, o `MinMaxScaler` é utilizado para colocar os valores de cada coluna da matriz de dados entre 0 e 1. Repare que objeto `scaler` é ajustado ao conjunto de treinamento.

```
In [9]: scaler = MinMaxScaler()
In [10]: scaler.fit(XTrain)
In [11]: XTrain = scaler.transform(XTrain)
```

Abaixo, 10 valores candidatos para o hiperparâmetro C são estabelecidos, os quais variam de 10^{-10} até 10^{10} em uma escala logarítmica.

```
In [12]: model = SVC(kernel = 'linear')
In [13]: cValues = np.logspace(-10, 10, 10)
In [14]: hParameters = {'C': cValues}
```

A classe `GridSearchCV` permite encontrar a melhor combinação de hiperparâmetros, neste caso apenas para C, por meio de busca exaustiva e validação cruzada.

```
In [15]: clf = GridSearchCV(model, hParameters, cv = 3)
In [16]: clf.fit(XTrain, yTrain)
```

Uma vez que a fase de treinamento é finalizada, é necessário testar o modelo no conjunto de testes. Para isso, é necessário primeiro aplicar a mesma transformação que foi aplicada ao conjunto de treinamento. Após isso, o método `predict` é utilizado para predizer a classe dos elementos do conjunto.

```
In [17]: XTest = scaler.transform(XTest)
In [18]: yPredicted = clf.predict(XTest)
```

Por fim, pode-se comparar a classe real dos objetos no conjunto de testes com as classes preditas. A seguir, um exemplo de como obter a matriz de confusão[13], *accuracy*, *precision*, *recall* e *F-Score*.

```
In [19]: confusion_matrix(yTest, yPredicted)
Out[19]:
array([[45,  2],
       [ 2, 65]])

In [20]: accuracy_score(yTest, yPredicted)
Out[20]: 0.9649122807017544

In [21]: precision_score(yTest, yPredicted)
Out[21]: 0.9701492537313433

In [22]: recall_score(yTest, yPredicted)
Out[22]: 0.9701492537313433

In [23]: f1_score(yTest, yPredicted)
Out[23]: 0.9701492537313433
```

Observe que o resultado final pode sofrer uma variação pequena, uma vez que o processo inicial de separar os conjuntos de treino e teste, bem como a validação cruzada, são feitos de forma aleatória. Alguns autores sugerem que todo o processo acima deve ser repetido algumas vezes para que a média e intervalos de confiança do desempenho do classificador no conjunto de teste também possam ser analisados.

Exemplo: Classificando Aplicações de Data Centers

Cada vez mais *data centers* e grandes *clusters* são utilizados para executar diversos tipos de aplicativos. Eles variam desde redes sociais e jogos até aplicativos com uso intensivo de computação, como indexação de conteúdo da Web, simulação de projetos de produtos, análise de dados provenientes de *crowdsourcing*. Outro tipo de software que demanda alto poder computacional são as aplicações científicas [Benson et al., 2010], como, por exemplo, previsão do tempo, previsão de desastres naturais, perfil bacteriano e genotipagem animal. Um aspecto interessante dessas aplicações é que a grande maioria apresenta padrões similares de computação e comunicação [Asanovic et al., 2006], o que significa que elas tendem a transmitir a mesma quantidade de informação entre os mesmos nós computacionais, independentemente dos dados de entrada.

Para melhorar a qualidade dos resultados, essas aplicações estão exigindo cada vez mais poder computacional, sendo executadas em sistemas de computacionais dedicados, demorando muitas horas para completar suas execuções e movimentando grandes volumes de dados, fazendo com que a rede se torne um gargalo. Então, entender as demandas de comunicação e classificar os aplicativos nos *data centers* é um desafio relevante [Srivastava et al., 2016]. A identificação correta de quais aplicativos estão usando os recursos do *data center* é essencial em várias atividades de gerenciamento, como planejamento de expansão, engenharia de tráfego, detecção de anomalias, monitoramento, e economia de energia [Trois et al., 2018].

[13]Na biblioteca *Scikit-learn*, as classes reais são as linhas e as preditas são as colunas

Existem três formas de classificar as aplicações. A primeira usa as portas de comunicação TCP para realizar essa tarefa; porém, como muitos aplicativos estão adotando a numeração dinâmica de portas, essa abordagem acaba sendo imprecisa. A segunda consiste em examinar o *payload* dos pacotes (*Deep Packet Inspection*) para classificar o tráfego. Essa abordagem não apenas impõe uma complexidade computacional significativamente maior, mas também requer conhecimento específico dos protocolos de aplicativos [Bujlow et al., 2013].

A terceira categoria aplica técnicas de aprendizado de máquina para classificar o tráfego. Algumas propostas exploram informações intrínsecas e estatísticas dos fluxo da rede para alimentar os classificadores [Eerman et al., 2006, Wang et al., 2010, Zhang et al., 2012, Fahad et al., 2014]; por exemplo, média e variação do tamanho do pacote, número total de pacotes ou bytes, duração do fluxo, números de porta do servidor e do cliente e muitos outros. Como aplicações de *data centers* apresentam padrões de comunicação [Asanovic et al., 2006], outra abordagem propõe extrair características de imagens geradas a partir da matriz de tráfego dos nodos computacionais [Trois et al., 2018].

Uma matriz de tráfego pode ser representada como uma matriz M_B, onde a entrada (i, j) de M_B contém o número de pacotes e/ou bytes transmitidos do nó i para o nó j. As matrizes de tráfego são normalizadas, formando uma imagem em tons de cinza, onde as células em preto são os pares de nós mais comunicativos e as células brancas indicam que nenhuma comunicação aconteceu. A Figura 6.19 mostra quatro exemplos de matrizes de tráfego coletadas de quatro aplicações científicas executadas em 16 nós computacionais.

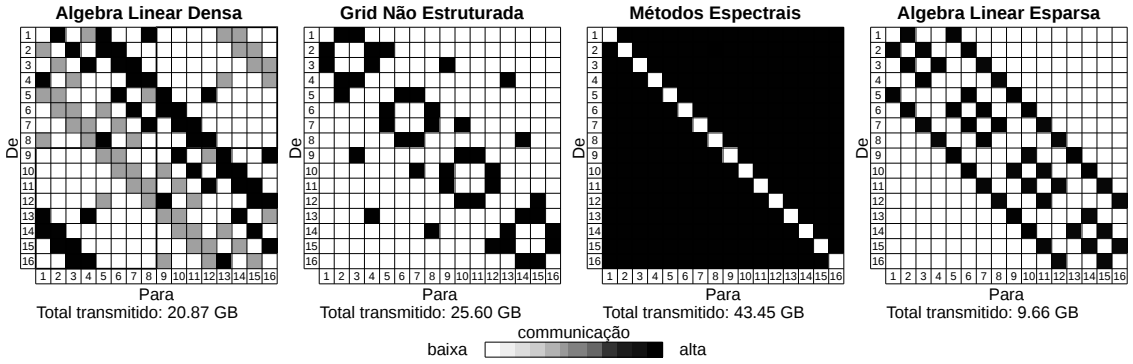


Figura 6.19: Matrizes de tráfego coletadas de 16 nós computacionais.

Para simplificar a implementação, as matrizes de tráfego serão transformadas em um vetor que será usado como vetor de característica do classificador[14]. As seções seguintes apresentam a leitura das imagens das matrizes de tráfego e a aplicação das funções de aprendizagem de máquina para reconhecer as aplicações.

[14]Será aplicada uma abordagem semelhante a este exemplo: `https://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html`

Lendo as Matrizes de Tráfego

As matrizes de tráfego usadas no exemplo apresentado nessa seção foram coletadas do BlueCrystal Fase 3[15], um ambiente computacional de alta performance pertencente à Universidade de Bristol. Ele é composto de 223 *blades*, onde cada *blade* tem processador SandyBridge de 2,6GHz com 16 núcleos, 64GB de RAM e um disco SATA de 1TB. Além desses, há também 100 *blades* com GPGPUs duplos e 18 com maior capacidade de memória, onde cada *blade* contém 256GB.

As aplicações científicas foram executadas usando 128 nós computacionais, sendo capturada uma matriz de tráfego por segundo, durante 30 minutos. Serão usadas matrizes de tráfego de quatro aplicações científicas[16] que simulam o escoamento de fluidos por meio do método de Lattice Boltzmann, implementadas usando a biblioteca OpenLB [Heuveline and Latt, 2007]. A Figura 6.20 mostra exemplos das matrizes coletadas.

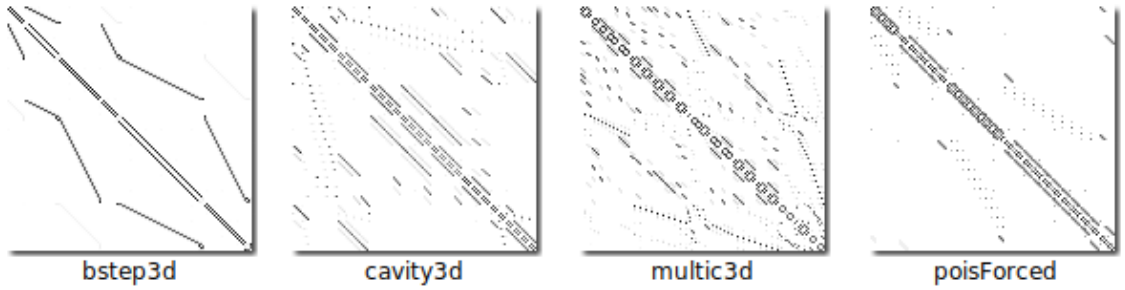


Figura 6.20: Matrizes de tráfego das aplicações científicas a serem classificadas.

É importante salientar que a maioria das aplicações possui várias fases de execução, por exemplo, fase de distribuição dos dados, fase de processamento e fase de sincronização. A Figura 6.21 mostra as matrizes de tráfego da aplicação científica `bstep3d`. Observa-se que nos tempos 380, 382 e 383, as matrizes de tráfego são semelhantes, pois representam a fase de processamento da aplicação. Por outro lado, no tempo 381 é feita a comunicação de sincronismo dos nodos, onde observa-se uma matriz bem distinta das demais.

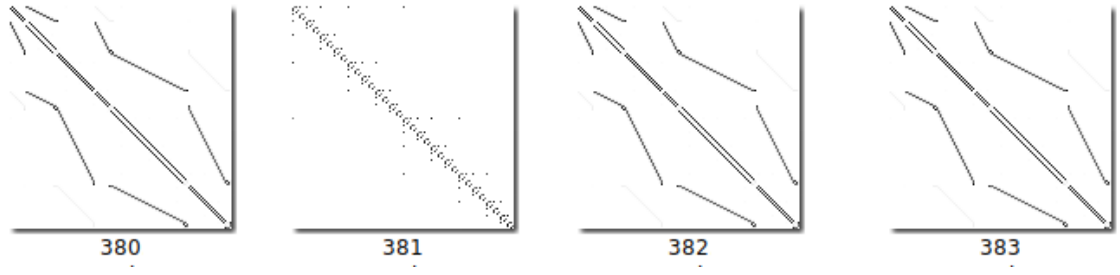


Figura 6.21: Matrizes de tráfego das fases de processamento e sincronização de uma aplicação científica.

[15]BlueCrystal Fase 3: `https://www.acrc.bris.ac.uk/acrc/phase3.htm`

[16]O dataset original, que apresenta 12 aplicações científicas além de diferentes fases de processamento do MapReduce, está disponível em: `www.inf.ufpr.br/ctrois/dctracs/dataset/`.

Para facilitar a implementação, as matrizes disponibilizadas neste exemplo foram filtradas, retirando-se as imagens que não correspondiam ao tráfego de processamento das aplicações. Primeiramente é criada uma lista com as aplicações que serão classificadas (linha 5); observe que o nome da aplicação deve ser igual ao diretório onde as fotos estão armazenadas.

```
In [1]: import glob # ler as imagens do diretorio
In [2]: import cv2 # converter a imagem em uma matriz python
In [3]: import numpy as np # converter a matriz em um vetor (reshape)

In [4]: # lista das aplicacoes que serão lidas
In [5]: aplicacoes=["bstep3d", "cavity3d", "multiComponent3d",
  ....:             "poiseuille3d"]
In [6]: # variavel auxiliar para nomear as classes
In [7]: classe=1

In [8]: # listas que armazenarão todas as imagens e respectivas classes
In [9]: matrizes=[]
In [10]: classes=[]
```

Em seguida, é feita a leitura das imagens. Foi usada a biblioteca glob, que retorna uma lista de todos os arquivos que estão dentro de um diretório. As imagens são lidas e transformadas para uma matriz Python pela função `cv2.imread`. Para usar os valores da matriz como vetor de características para o classificador, é necessário transformá-la em um vetor; isso é feito através da função `np.reshape` (linha 16).

```
In [11]: # percorre todas as aplicações
In [12]: for a in aplicacoes:
In [13]:     # percorre todas as imagens do diretorio
In [14]:     for i in glob.glob("./imagens/"+a+"/*"):

In [15]:         # le a imagem, converte para vetor e adiciona na lista
In [16]:         matrizes.append(np.reshape(cv2.imread(i,0),-1))
In [17]:         # inclui o numero da classe na lista de classes
In [18]:         classes.append(classe)

In [19]:     # proxima aplicacao: incrementa o numero da classe
In [20]:     classe+=1
```

Classificando as Aplicações

Uma vez que as imagens foram lidas e transformadas em um vetor (linha 16) e suas respectivas classes foram armazenadas (linha 18), serão aplicadas as funções de aprendizagem de máquina. Primeiramente será aplicada a função `train_test_split` para dividir as imagens aleatoriamente em dois conjuntos, 50% para treino e 50% para teste (parâmetro `test_size=0.5`).

```
In [21]:  from sklearn.model_selection import train_test_split
In [22]:  X_train, X_test, y_train, y_test = train_test_split(matrizes,
   ....:                                          classes, test_size=0.5)
```

Então, é criada uma instância do classificador `RandomForestClassifier` (linha 24) e o seu treinamento é realizado (linha 26).

```
In [23]: from sklearn.ensemble import RandomForestClassifier
In [24]: classifier = RandomForestClassifier(n_estimators=100,
   ....:                                     max_depth=2,random_state=0)
In [25]: # treina o classificador
In [26]: classifier.fit(X_train, y_train)
```

Finalmente é feita a classificação dos dados de teste `X_test` (linha 29) e o relatório de classificação e a matriz de confusão são apresentados (linhas 31 e 33).

```
In [27]: from sklearn import metrics

In [28]: expected = y_test
In [29]: predicted = classifier.predict(X_test)

In [30]: print("Classification report:\n%s\n"%
   ....:         (metrics.classification_report(expected, predicted)))
Out[31]:
Classification report:
             precision    recall  f1-score   support

          1       0.00      0.00      0.00       748
          2       1.00      1.00      1.00       836
          3       0.48      1.00      0.65       704
          4       1.00      1.00      1.00       857

avg / total       0.65      0.76      0.68      3145

In [32]: print("Confusion matrix:\n%s" %
   ....:       metrics.confusion_matrix(expected, predicted))
Out[33]:
Confusion matrix:
[[  0   0 748   0]
 [  1 835   0   0]
 [  0   1 703   0]
 [  0   0   0 857]]
```

Neste caso de estudo, foi apresentada uma forma de classificar aplicações usando suas matrizes de tráfego como entrada para algoritmos de aprendizagem de máquina. Conforme assertado por Trois et al. [Trois et al., 2018], os métodos existentes que aplicam aprendizagem de máquina para classificar aplicações usam o mesmo tipo de representação (estatísticas de fluxos), então existe espaço de pesquisa para investigar outras informações que permitam discriminar os diferentes tipos de tráfego de rede.

Algoritmos de aprendizagem supervisionada foram apresentados nessa parte do minicurso. Como exemplo, foi reportado uma forma de classificar as aplicações de centro de dados através da identificação dos padrões expressos nas matrizes de comunicação de seus nodos computacionais.

Conclusões e Perspectivas

O mercado profissional sofre modificações constantemente em função principalmente das novas demandas surgidas na sociedade e das novas tecnologias que vão surgindo. Esse contexto fez surgir uma nova profissão: o cientista de dados. A ciência de dados, também chamada de *data science*, é um reflexo do ambiente interconectado no qual vivemos e da enorme quantidade de dados disponibilizados e trafegados nas redes sociais e nas redes de computadores. A ciência de dados é uma área interdisciplinar que aborda as áreas de ciências exatas e engenharia, tais como programação, matemática e estatística.

No contexto das redes de computadores, o conhecimento da área de ciência de dados pode ajudar os administradores e gerentes de rede na tomada de decisões como implementação de soluções de engenharia de tráfego, planejamento de capacidade, detecção de tendências e anormalidades. Este minicurso teve como objetivo principal habilitar o profissional de redes com as competências básicas de um profissional da área de ciência de dados a partir de exemplos práticos contextualizados na área.

A abordagem adotada neste minicurso serviu para mostrar que, apesar de complexas, muitas técnicas podem ser utilizadas de forma simples por meio da linguagem de programação Python e algumas de suas bibliotecas. Foram abordados aspectos relacionados à análise exploratória de dados, com conteúdos de leitura, manipulação e visualização de dados e redução de dimensionalidade. Em seguida foi apresentado o conceito de aprendizado não supervisionado com exemplos de técnicas de agrupamento e detecção de comunidades. Por fim, o aprendizado supervisionado foi apresentado por meio de exemplos de diversas técnicas de classificação. A metodologia tradicional, que consiste em treino, validação e teste foi explicitada e contextualizada com exemplos práticos que podem ser aplicado no dia a dia do profissional de redes de computadores. Como resultado, espera-se que este minicurso tenha contribuído para capacitar os administradores, gerentes e pesquisadores de redes de computadores com o ferramental básico para realização de análise de dados a partir de suas redes e, com isso, essa informação auxilie suas tomadas de decisão e pesquisas.

Em se tratando de pesquisas acadêmicas e perspectivas futuras, a literatura traz diversos exemplos de como a ciência de dados pode ser aplicada em redes. Com o objetivo de mostrar como os resultados desse "casamento" pode auxiliar tomadas de decisão, reduzir custos e melhorar o desempenho das redes de computadores, em [Boutaba et al., 2018] é apresentado um apanhado bastante completo de como as técnicas apresentadas neste minicurso, aliadas às técnicas de aprendizado de máquina podem vêm sendo utilizadas na solução de diferentes desafios relacionados às redes de computadores.

Para motivar o leitor interessado em aprofundar-se na área e treinar seus conhecimentos com bases de dados reconhecidas e referenciadas, recomenda-se visitar o sítio eletrônico do CAIDA[17] (*Center for Applied Internet Data Analysis*) que realiza a coleta, curadoria e compartilhamento de dados para análise científica de tráfego na Internet, topologia, roteamento, desempenho e eventos relacionados à segurança. Todos os dados disponíveis no CAIDA, possuem *links* para descrições, formulários de solicitação de dados restritos, relatórios em tempo real e outros metadados.

[17] https://www.caida.org/data/

Agradecimentos

Este trabalho recebeu financiamento parcial do projeto Horizon 2020 (União Européia) sob no. 688941 (FUTEBOL), assim como do MCTI por meio da RNP e do CTIC. Além disso, este estudo foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001, além de financiamento parcial proveniente de recursos de bolsas e projetos CNPq, FAPES e FAPEMIG. Os autores agradecem também à UFSM/FATEC pelo finaciamento parcial, através do projeto 041250 - 9.07.0025 (100548).

Referências

[Asanovic et al., 2006] Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., Patterson, D. A., Plishker, W. L., Shalf, J., Williams, S. W., et al. (2006). The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

[Benson et al., 2010] Benson, T., Anand, A., Akella, A., and Zhang, M. (2010). Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99.

[Boutaba et al., 2018] Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., and Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16.

[Bujlow et al., 2013] Bujlow, T., Carela-Español, V., and Barlet-Ros, P. (2013). Comparison of deep packet inspection (dpi) tools for traffic classification. Technical report, Universitat Politècnica de Catalunya.

[Carvalho et al., 2014] Carvalho, T., Junior, Marcos Simplicio, R. F. R. M., and Magri, D. (2014). Dmz científica: Desafios e modelos de gerenciamento de aplicações de alto volume de dados no contexto de e-ciência. In *SBRC 2014 - Minicursos*.

[Celes et al., 2017] Celes, C., De Oliveira Nunes, I., Borges, J., Silva, F., De Abreu COtta, L., Vaz de Melo, P., Ramos, H., Andrade, R., and Loureiro, A. (2017). *Big Data Analytics no Projeto de Redes Móveis: Modelos, Protocolos e Aplicações*, pages 1–58.

[Eerman et al., 2006] Eerman, J., Mahanti, A., and Arlitt, M. (2006). Internet traffic identification using machine learning techniques. In *Proc. of the 49th IEEE Global Telecomm. Conf.(GLOBECOM)*, pages 1–6.

[Fahad et al., 2014] Fahad, A., Tari, Z., Khalil, I., Almalawi, A., and Zomaya, A. Y. (2014). An optimal and stable feature selection approach for traffic classification based on multi-criterion fusion. *Future Generation Computer Systems*, 36:156–169.

[Forgy, 1965] Forgy, E. W. (1965). Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications. *Biometrics*, 21:768–769.

[Harari, 2016] Harari, Y. N. (2016). *Homo Deus: A brief history of tomorrow*. Random House.

[Heuveline and Latt, 2007] Heuveline, V. and Latt, J. (2007). The openlb project: an open source and object oriented implementation of lattice boltzmann methods. *International Journal of Modern Physics C*, 18(04):627–634.

[Knight et al., 2011] Knight, S., Nguyen, H. X., Falkner, N., Bowden, R., and Roughan, M. (2011). The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775.

[Kurzweil, 2010] Kurzweil, R. (2010). *The singularity is near*. Gerald Duckworth & Co.

[Lakhina et al., 2005] Lakhina, A., Crovella, M., and Diot, C. (2005). Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4):217–228.

[McKinney, 2013] McKinney, W. (2013). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. O'Reilly, Beijing, first edition.

[Metsis et al., 2006] Metsis, V., Androutsopoulos, I., and Paliouras, G. (2006). Spam filtering with naive bayes-which naive bayes?

[Roy and Vetterli, 2007] Roy, O. and Vetterli, M. (2007). The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pages 606–610.

[Srivastava et al., 2016] Srivastava, G., Singh, M., Kumar, P., and Singh, J. (2016). Internet traffic classification: A survey. In *Recent Advances in Mathematics, Statistics and Computer Science*, pages 611–620. World Scientific.

[Trois et al., 2018] Trois, C., Bona, L. C., Oliveira, L. S., Martinello, M., Harewood-Gill, D., Fabro, M. D. D., Nejabati, R., Simeonidou, D., Lima, J. C. D., and Stein, B. (2018). Exploring textures in traffic matrices to classify data center communications. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 1123–1130.

[Trois et al., 2016] Trois, C., Fabro, M. D. D., de Bona, L. C. E., and Martinello, M. (2016). A survey on sdn programming languages: Toward a taxonomy. *IEEE Communications Surveys Tutorials*, 18(4):2687–2712.

[Wang et al., 2010] Wang, Y., Xiang, Y., and Yu, S. Z. (2010). Automatic application signature construction from unknown traffic. In *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 1115–1120.

[Zaki and Jr, 2014] Zaki, M. J. and Jr, W. M. (2014). *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, New York, NY, USA.

[Zhang et al., 2012] Zhang, H., Lu, G., Qassrawi, M. T., Zhang, Y., and Yu, X. (2012). Feature selection for optimizing traffic classification. *Computer Communications*, 35(12):1457–1471.

Realização

Organização

Apoio Institucional

Patrocínio

Diamante

Ouro

Prata

Bronze