ERRC 2019

Alegrete - RS
16 a 19 de setembro

17ª Escola Regional de Redes de Computadores

Minicursos da XVII Escola Regional de Redes de Computadores

Organização e Edição:
Diego Kreutz
Rodrigo B. Mansilha
Charles C. Miers

Realização

SBC Sociedade Brasileira de Computação

Organização

unipampa Universidade Federal do Pampa

PPGES Programa de Pós-Graduação em Eng. de Software

PampaTec parque tecnológico do pampa

INSTITUTO FEDERAL Farroupilha

Patrocínio Diamante

Sicredi

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico

Ouro

RCT redeconesul Telecom

interneith

Prata

AVATO Inteligência em Conectividade e TI

ACESSO INFORMÁTICA

TREND MICRO

Bronze

Parceria Clube de Tiro Indoor

17º Escola Regional de Redes de Computadores

ERRC 2019

16 a 19 de setembro de 2019
Alegrete-RS, Brasil

MINICURSOS

Editora
Sociedade Brasileira de Computação - SBC
Porto Alegre - RS

Organizadores
Diego Kreutz
Rodrigo Brandão Mansilha
Charles Christian Miers

Realização
Universidade Federal do Pampa (UNIPAMPA)

Patrocínio Governamental
CNPq

Patrocínio Empresarial
Sicredi, RCT RedeConesul Telecom, Interneith, Ávato,
Acesso Informática, Trend Micro

Promoção
Sociedade Brasileira de Computação - SBC
Comissão Especial de Redes de Computadores e Sistemas Distribuídos
Secretaria Regional do Rio Grande do Sul

Capa: Camila Martini
Supervisão Gráfica: Diego Kreutz, Rodrigo B. Mansilha e Charles C. Miers

Dados Internacionais de Catalogação na Publicação (CIP)

E74

Escola Regional de Redes de Computadores (17.: 2019: Alegrete-RS, RS)

Minicursos da XVII Escola Regional de Redes de Computador [recurso eletrônico] – Organizadores: Diego Kreutz, Rodrigo B. Mansilha, Charles C. Miers – Porto Alegre : SBC, 2020.

ISBN 978-65-87003-29-0

1. Computação - Congresso. 2. Rede de computadores. 3. Segurança da informação. I. Kreutz, Diego. II. Mansilha, Rodrigo B. III. Miers, Charles C. IV. Sociedade Brasileira de Computação. V. Universidade Federal do Pampa. VI. Universidade do Estado de Santa Catarina. VII. Título.

CDU004

Catalogação elaborada por Francine Conde Cabral
CRB-10/2606

Apresentação

A Escola Regional de Redes de Computadores (ERRC 2019), em sua XVII edição, ocorreu em Alegrete, no estado do Rio Grande do Sul (RS), nos dias 16 a 19 de setembro de 2019. A ERRC é promovida pela Sociedade Brasileira de Computação (SBC). Seu objetivo é promover a pesquisa e qualificar estudantes e profissionais nas áreas de Redes de Computadores, Sistemas Distribuídos e Segurança da Informação. Com relação à Segurança da Informação, é importante destacar o Workshop Regional de Segurança da Informação e de Sistemas Computacionais (WRSeg 2019), parte integrante da ERRC, que está em sua quarta edição.

A programação da ERRC e do WRSeg 2019 compreende cinco minicursos, vinte e seis artigos, oito resumos estendidos, quatro palestras e sete sessões técnicas, sendo três do WRSeg. Neste livro de anais encontram-se apenas os textos dos minicursos da edição de 2019 da ERRC. Os artigos das sessões técnicas estão disponíveis através da OpenLib da SBC.

Na edição deste ano, organizada pela Universidade Federal do Pampa (Unipampa) e pelo Instituto Federal Farroupilha (IFFarroupilha), tivemos o patrocínio do CNPq, Sicredi, RCT RedeConesul Telecom, Interneith, Ávato, Acesso Informática, Trend Micro, Clube de Tiro Parceria Indoor e o apoio da Raptor - Engenharia e Inovação Tecnológica e Garupa. Agradecemos aqui a todos os patrocinadores e colaboradores da organização.

Expressamos aqui também os nossos mais sinceros agradecimentos a todos os autores dos minicursos, palestrantes, autores e apresentadores de trabalhos técnicos, que contribuíram para o sucesso da ERRC 2019! Também agradecemos imensamente aos revisores de trabalhos da ERRC e do WRSeg, que contribuíram para qualificar os trabalhos da escola. Vários autores manifestaram explicitamente o contentamento com a qualidade das revisões realizadas.

Diego, Rodrigo e Charles
Organizadores dos Anais de Minicursos da ERRC
Alegrete-RS, Brasil, setembro de 2019.

ERRC 2020

17º Escola Regional de Redes de Computadores

Comitê Organizador

Coordenação Geral
Rodrigo Brandão Mansilha (UNIPAMPA)
Marcelo Caggiani Luizelli (UNIPAMPA)

Coordenação Local
Arthur F. Lorenzon (UNIPAMPA)
Cláudio Schepke (UNIPAMPA)
Diego Kreutz (UNIPAMPA)
Fábio Rossi (IFFarroupilha)

Coordenação (colaboração)
Fauzi Shubeita (Setrem)
Charles Christian Miers (UDESC)

Organização do Comitê de Programa da ERRC
Diego Kreutz (UNIPAMPA)
Ricardo Schimidt (UPF)

Organização Comitê de Programa WRSeg
Tiago Rizzetti (CTISM/UFSM)

Organização de Minicursos e Oficinas
Leonardo Pinho (UNIPAMPA)
Jéferson Nobre (UFRGS)
Guilherme Koslovski (UDESC)
Ricardo Luis dos Santos (IFRS)

Organização e Captação de Patrocínios
Magnum Foletto (Ávato)
Fauzi Shubeita (SETREM)
Gerson Battisti (Unijui)

Maratonas de Programação (Redes e Segurança)
Roger Immich (IC/UNICAMP)
Rodrigo Righi (Unisinos)
Tiago Rizzetti (CTISM/UFSM)

Organização de Caravanas
Sandro Lemos Oliveira (UNIPAMPA)

Comitê de Programa da ERRC

Adenauer Yamin (UFPEL)
Alberto Schaeffer-Filho (UFRGS)
Andrea Charao (UFSM)
André Riker (UFPA)
Antonio Candia (UFSM)
Arthur Lorenzon (UNIPAMPA)
Bruno Dalmazo (UFRGS)
Bruno Vizzotto (UNIPAMPA)
Carlos Raniery Paula dos Santos (UFSM)
Carlos Vinícius Rasch Alves (SENAC-RS)
Charles Christian Miers (UDESC)
Clarissa Marquezan (Huawei)
Claudio Schepke (UNIPAMPA)
Cristiano Both (UNISINOS)
Cristina Nunes (PUC-RS)
César Loureiro (IFRS-Porto Alegre)
Dalvan Griebler (SETREM)
Dartagnan Dias de Farias (SENAC-RS)
Diego Kreutz (UNIPAMPA)
Diogo Menezes Ferrazani Mattos (UFF)
Djamel Sadok (UFPE)
Douglas Macedo (UFSC)
Edison Pignaton De Freitas (UFRGS)
Eduardo Moroñas Monks (SENAC-RS/UFPEL)
Erico Amaral (UNIPAMPA)
Ewerton Salvador (UFPB)
Felipe Nunes (AMF)
Fábio Rossi (PUC-RS)
Gerson Batistti (UNIJUI)
Gerson Soares (UNIMI)
Giani Petri (UFSM)
Glederson Santos (IFSul)
Guilherme Koslowski (UDESC)
Guilherme Sperb Machado (UZH)
Juliano Wickboldt (UFRGS)
Leandro Bertholdo (UFRGS)
Leonardo Pinho (UNIPAMPA)
Lisandro Zambenedetti Granville (UFRGS)
Lucas Bondan (UFRGS)
Lucas F. Müller (UFRGS)
Luciano Ignaczak (UNISINOS)
Luciano Paschoal Gaspary (UFRGS)
Lucio Prade (UNISINOS)
Luis Augusto Dias Knob (IFRS-Sertão)
Marcelo Luizelli (UNIPAMPA)
Marcia Pasin (UFSM)
Marco Antonio Torrez Rojas (IFSC)
Marco Aurélio Spohn (UFFS)
Matias Schimuneck (UFRGS)
Pedro Marcos (FURG)
Pedro Sá da Costa (Copelabs/PT)
Rafael Bastos (UFPEL)
Rafael Esteves (IFRS)
Rafael Kunst (UNISINOS)
Raul Ceretta Nunes (UFSM)
Regio Michelin (IFRS)
Reinaldo Gomes (UFCG)
Ricardo José Pfitscher (UFRGS)
Ricardo Luis dos Santos (UFRGS)
Ricardo Schmidt (UPF)
Roben Lunardi (IFRS)
Rodolfo Antunes (UNISINOS)
Rodrigo Calheiros (Western Sydney University)
Rodrigo Mansilha (UNIPAMPA)
Rodrigo Righi (UNISINOS)
Roger Immich (IC/UNICAMP)
Rogério Turchetti (UFSM)
Simone Ceolin (UFSM)
Tiago Ferreto (PUC-RS)
Valter Roesler (UFRGS)
Vinicius Vielmo Cogo (ULisboa)
Vinícius Guimarães (IFSUL)

Comitê de Programa do WRSeg

Adenauer Yamin (UFPEL)
Bruno Dalmazo (UFRGS)
Bruno Mozzaquatro (UFSM)
Carlos Vinicius Rasch Alves (SENAC-RS)
Charles Christian Miers (UDESC)
Diego Kreutz (UNIPAMPA)
Diogo Menezes Ferrazani Mattos (UFF)
Eduardo Monks (SENAC-RS)
Erico Hoff do Amaral (UNIPAMPA)
Felipe Nunes (AMF)
Luciano Ignaczak (UNISINOS)
Luis Augusto Dias Knob (IFRS-Sertão)
Marco Antônio Sandini Trentin (UPF)
Márcia Henke (UFSM)
Reinaldo Gomes (UFCG)
Ricardo Almeida (UFPEL)
Ricardo Schmidt (UPF)
Roben Castagna Lunardi (IFRS-Restinga)
Roberto Samarone Araújo (UFPA)
Rodrigo Mansilha (UNIPAMPA)
Rogério Turchetti (UFSM)
Tiago Antônio Rizzetti (UFSM)
Walter Priesnitz Filho (UFSM)

SBC

Sociedade Brasileira de Computação

Secretaria Regional do Rio Grande do Sul
Jean Felipe Patikowski Cheiran (UNIPAMPA) - Secretário

Secretaria Regional de Santa Catarina
Everaldo Artur Grahl (FURB)- Secretário

Secretaria Regional do Paraná
Thelma Elita Colanzi (UEM)- Secretária

CRRC

Comitê Assessor da Escola Regional de Redes de Computadores do Rio Grande do Sul

Instituição	Representante
UFSM	Carlos Raniery
UFRGS	Jéferson Nobre
UFRGS	Juliano Wickboldt
IFRS/Sertão	Luis Knob
SENAC/POA	Marcelo Conterato
UFSM	Márcia Pasin
UNIPAMPA	Rodrigo Brandão Mansilha
PUCRS	Tiago Ferreto

Sumário

Capítulo 1

Introdução aos Blockchains: Teoria e Prática

João Otávio Chervinski, Felipe Melchior, Rafael Fernandes, Guilherme Neri, Lucas Antunes, Diego Kreutz, Rodrigo Mansilha (Unipampa)

O lançamento de criptomoedas como a Bitcoin[1], Monero[2] e Ripple[3] deu origem a uma revolução nos sistemas de pagamento no mundo todo. A Bitcoin é o primeiro e mais conhecido caso disruptivo de sucesso. Criptomoedas como a Bitcoin funcionam de maneira descentralizada, isto é, sem a necessidade de uma autoridade reguladora, permitindo que qualquer um participe da rede e efetue transações.

Em sistemas financeiros convencionais, quando pagamentos são efetuados através de uma empresa de cartões de crédito, o usuário estabelece uma relação de confiança com a empresa e delega a ela a responsabilidade de validar os seus dados e efetuar a transação corretamente. Diferentemente dos sistemas tradicionais, uma das principais inovações da Bitcoin foi a criação de um sistema descentralizado no qual participantes são capazes de enviar pagamentos uns aos outros sem a necessidade de estabelecer confiança prévia. Em outras palavras, criptomoedas como a Bitcoin possuem um grande apelo pelo fato de eliminarem a necessidade de uma autoridade intermediária. Isto é realizado através da tecnologia de *blockchain*.

Um *blockchain* é uma estrutura de dados distribuída, formada por uma série de blocos de informação encadeados. Cada participante da rede pode obter uma cópia completa dos dados e compartilhá-la com outros participantes. Numa rede de *blockchain*, os usuários trabalham de maneira colaborativa para validar transações, utilizando criptografia para garantir a sua segurança e verificabilidade. Mecanismos criptográficos também são utilizados para garantir a ordem dos blocos e evitar a sua adulteração, visto que só deve existir uma sequência válida de blocos. Entretanto, apesar dos *blockchains* serem utilizados principalmente em criptomoedas, já existem propostas e exemplos práticos de aplicação dessa tecnologia em áreas e temas como sistemas financeiros, registro e proteção de propriedade intelectual, sistemas baseados em reputação, cadeia de suprimentos, setor de energia, setor de saúde, e-gov, marketing, sistemas de votação eletrônica, aplicações industriais, sistemas inteligen-

[1] https://bitcoin.org
[2] https://www.getmonero.org
[3] https://ripple.com

tes, realidade aumentada e jogos [Zheng et al. 2018, Hoy 2017, Olnes et al. 2017, Chen et al. 2018, Marinoff 2018, Dai et al. 2019, Al-Jaroodi and Mohamed 2019, Abbas and Sung-Bong 2019, Min et al. 2019, Yang et al. 2019, Xie et al. 2019].

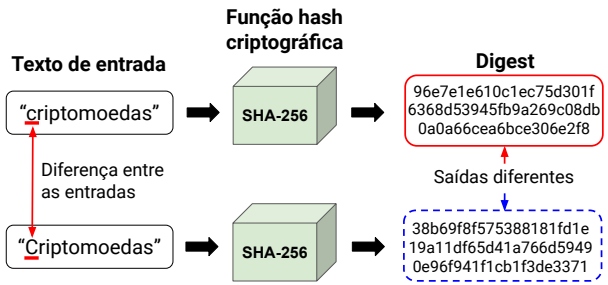
Este minicurso tem como objetivo apresentar os principais conceitos inerentes às tecnologias de *blockchain* e criptomoedas, contribuindo para a difusão do conhecimento desta área que está no seu auge de pesquisa e desenvolvimento. No decorrer do minicurso são abordadas também questões relacionadas à segurança (e.g., integridade de dados) em criptomoedas como a Bitcoin. Por fim, vale ressaltar que atualmente há muitas oportunidades, tanto na academia (exemplo: novas conferências específicas, criadas em 2018 e 2019) quanto no mercado (exemplo: muitas empresas investindo e apostando nessas novas tecnologias), relacionadas diretamente aos temas discutidos aqui.

O restante do minicurso está dividido como segue. Na Seção 1.1 são apresentados e discutidos os detalhes da tecnologia *blockchain*. A Seção 1.2 discute as criptomoedas e métodos de mineração e criação de novas moedas. O funcionamento da criptomoeda Bitcoin é detalhado na Seção 1.3. Nas Seções 1.4 e 1.5 são apresentadas o sistema de simulação de *blockchains* UniBlock e as considerações finais do minicurso, respectivamente.

1.1. Blockchain

O *blockchain* é um registro de informações distribuído formado por uma cadeia de blocos de dados, conectados uns aos outros por um sistema que utiliza funções *hash* criptográficas. A Figura 1.1 apresenta o resultado da aplicação da função *hash* criptográfica SHA-256 (Secure Hash Algorithm-2) [NIST 2018] em duas entradas distintas. Como pode ser observado, a saída da função é completamente diferente para as duas entradas aparentemente idênticas. A utilização desse tipo de função é essencial para garantir a segurança e a integridade dos dados presentes no *blockchain*.

Figura 1.1: Aplicação de uma função *hash* criptográfica.



Funções *hash* tradicionais transformam dados de entrada de tamanho arbitrário em uma saída de tamanho fixo, chamada de *digest* ou *hash*. Funções *hash* criptográficas são um tipo especial de funções *hash* que devem possuir as seguintes propriedades:

(p_1) Um mesmo dado de entrada da função deve sempre retornar a mesma saída.

(p_2) Computar um *digest* para uma determinada entrada deve ser computacionalmente fácil, i.e., a operação pode ser realizada em menos de um segundo.

(p_3) Descobrir quais dados foram utilizados como entrada da função, analisando somente o *digest*, deve ser computacionalmente muito difícil. Uma tentativa por força bruta para descobrir os dados de entrada deve necessitar de vários anos considerando as capacidades atuais de processamento.

(p_4) Encontrar duas entradas que gerem o mesmo *digest* deve ser computacionalmente muito difícil, necessitando de vários anos de processamento.

(p_5) Uma pequena mudança na entrada deve alterar o *digest* resultante de tal forma que não seja possível encontrar alguma co-relação entre as saídas geradas pelo dado original e pelo dado alterado.

Este tipo de função é utilizado para ajudar a garantir a integridade de informações, já que qualquer mudança nos dados de entrada (e.g., um único bit, um único byte) altera o *digest* resultante, como pode ser observado na Figura 1.1. Ao aplicar uma função *hash* criptográfica em um arquivo e enviar o *digest* para a pessoa que irá recebê-lo, o receptor poderá aplicar a função novamente sobre o arquivo e verificar se o resultado é igual ao *digest* recebido. Dessa forma, ela garante que os dados não foram modificados, desde que o *digest* tenha sido recebido de forma segura.

Por exemplo, suponha que João trabalhe no setor financeiro de uma empresa e que tenha sido encarregado de realizar uma transferência de dinheiro da conta da empresa para a conta de algumas empresas parceiras. João recebeu de Maria o arquivo contendo os dados das contas bancárias por email. Um usuário malicioso, realizando um ataque de interceptação de dados, pode alterar o documento contido no email antes que ele seja recebido por João, adicionando novas contas bancárias na lista, por exemplo. Para certificar-se de que João receberá o arquivo com as mesmas informações enviadas originalmente, Maria computa o *digest* do documento anexado no email. Considere que a empresa utiliza um canal seguro para comunicação auxiliar, para o envio de pequenas quantidades de dados como os *digests*. Maria envia o *digest* do arquivo para João utilizando o canal de comunicação seguro. Ao fazer o *download* do arquivo com os dados das contas, João precisa certificar-se de que nada foi modificado. Para isso, ele computa o *digest* do arquivo recebido e compara-o com o *digest* enviado por Maria. Se os *digests* forem iguais, o documento não foi modificado.

No *blockchain*, a conexão lógica entre os blocos de dados é estabelecida e garantida através de *digests* de uma função *hash* criptográfica. Cada bloco na cadeia aponta para um bloco anterior identificado por um *digest* único (*digest* do bloco). Isso fornece uma propriedade interessante para o registro de transações: quando um bloco é adicionado ao final da cadeia de informações, torna-se uma tarefa muito difícil alterá-lo. Suponha a existência de um *blockchain* para uma aplicação bancária fictícia. Cada bloco da aplicação armazena a informação de uma transação entre duas contas. Um exemplo de um conjunto de dados armazenados em cada bloco de um *blockchain* fictício voltado para o armazenamento de transações financeiras é:

(d_1) *Versão*: Versão do sistema utilizada.

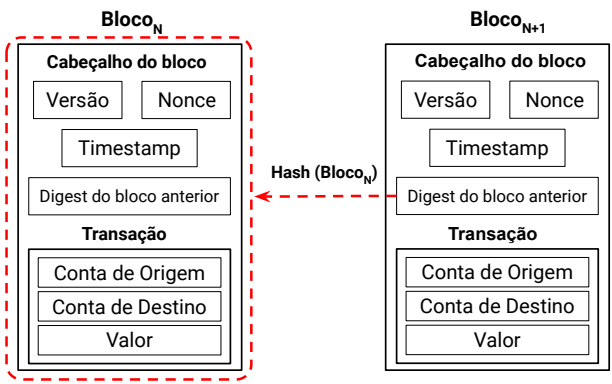
(d_2) *Timestamp*: Representação da data e hora de criação do bloco.

(d_3) *Nonce*: Número auxiliar utilizado para realização do cálculo da função *hash* criptográfica.

(d_4) *Digest* do bloco anterior: Resultado da aplicação de uma função *hash* criptográfica sobre os dados do bloco anterior.
(d_5) *Conta de origem*: A conta de onde o dinheiro será retirado.
(d_6) *Conta de destino*: A conta que receberá o dinheiro.
(d_7) *Valor da transação*: Quantia de dinheiro a ser transferida.

A Figura 1.2 ilustra a conexão entre uma cadeia de blocos. Considerando a estrutura de bloco apresentada, a informação que garante a ordem correta dos blocos é o ponteiro para o *digest* do bloco anterior na cadeia. A utilização de uma função *hash* criptográfica nos dados de cada bloco gera um *digest* único, permitindo estabelecer um vínculo forte e único entre os dados. Isso garante que exista uma única sequência válida de blocos.

Figura 1.2: Representação dos blocos em um *blockchain*.



Imagine que um atacante deseja modificar o conteúdo de um bloco para fins maliciosos, como direcionar o valor de uma transação para si mesmo. Para isso, será necessário que os novos dados do bloco gerem um *digest* igual ao anterior; caso contrário, a ligação entre a cadeia de blocos será desfeita. Se a cadeia de blocos for desfeita, o *blockchain* ficará em um estado inconsistente.

Como as funções *hash* criptográficas garantem que a probabilidade de gerar o mesmo *digest* a partir de dados de entrada diferentes é extremamente baixa, seria mais fácil alterar também o campo que aponta para o *digest* do bloco anterior nos blocos seguintes. Bastaria que o atacante mudasse os campos nos blocos posteriores, calculasse seus *digests* e repetisse o processo alterando os ponteiros até o final do *blockchain*. No entanto, um dos sistemas utilizado para controlar a criação de novos blocos, chamado de Prova de Trabalho, do inglês *Proof-of-Work* (PoW), evita que isso aconteça. Através desse sistema, não basta apenas calcular o *digest* do bloco para que ele seja adicionado à cadeia. O resultado da função *hash* deve obedecer a uma restrição que requer um grande esforço computacional para ser atendida. A restrição adotada na prática pela criptomoeda Bitcoin é encontrar um bloco cujo *digest* resultante possua os primeiros n bits iguais a zero, onde n depende da dificuldade de mineração determinada pelo sistema. Para variar

a saída da função *hash* e encontrar um bloco que atenda a restrição é necessário alterar os dados de entrada, para esta finalidade há em cada bloco um campo chamado de *nonce*, que é alterado até que o resultado desejado seja obtido.

Um usuário que deseja criar um *digest* válido altera os dados do campo *nonce* até que a execução da função *hash* gere o resultado desejado. Como não é possível prever o resultado da aplicação de uma função *hash* criptográfica, para cumprir o desafio, quem deseja criar um bloco válido deve tentar valores diferentes no campo *nonce* até que o *digest* resultante do bloco atenda às exigências do sistema. Após descoberto o *nonce* que torna o bloco válido, calcular novamente o *digest* do bloco torna-se trivial. Dois blocos diferentes podem possuir o mesmo dado no campo *nonce*, porém, os dados de transações não podem ser iguais. Isso faz com que *digests* iguais não possam ser gerados a partir de blocos diferentes.

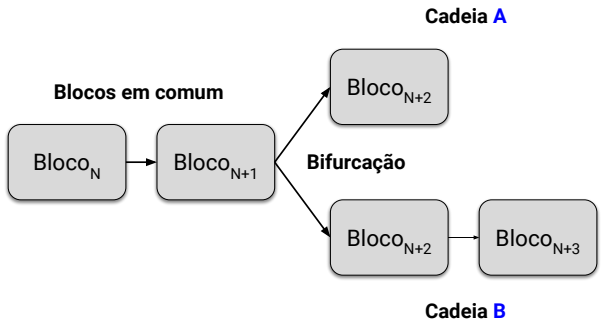
A descentralização do *blockchain* também dificulta a execução do ataque de modificação de blocos, pois alterações na cadeia de blocos implicam em mudar todas as outras cópias do *blockchain*, armazenadas por outros usuários. Para que um atacante possa controlar a criação de novos blocos, ele deve possuir mais de 50% do poder computacional de toda a rede. Somente assim seria possível criar blocos válidos com uma velocidade maior do que o resto dos usuários. Além do PoW, foram propostos outros esquemas de criação de blocos, como o *Proof-of-Stake* (PoS), o *Proof-of-Activity* (PoA) e o *Proof-of-Publication* (PoP) [Tschorsch and Scheuermann 2016].

Eventualmente, pode ocorrer uma bifurcação no *blockchain* quando novos blocos são adicionados, causada pelo tempo necessário para a propagação dos blocos na rede. A Figura 1.3 ilustra o estado da cadeia de blocos quando ocorre uma bifurcação. O problema ocorre quando um usuário *A* cria um um bloco válido ao mesmo tempo em que um usuário *B* cria outro bloco válido contendo dados diferentes. Como é necessária uma quantia de tempo para que a informação seja propagada aos outros participantes da rede, aqueles que recebem primeiro o bloco de *A*, o colocam no fim de suas cadeias, enquanto que aqueles que recebem primeiro o bloco de *B*, colocam um bloco no final de suas cadeias diferente dos primeiros. Isso causa uma divergência momentânea no *blockchain*. A partir desse momento existem duas cadeias cuja única diferença são os últimos blocos. Não é possível saber qual das cadeias é a correta.

Para resolver o problema da bifurcação na cadeia de blocos, alguns sistemas empregam uma estratégia chamada de “a regra da cadeia mais longa”. Com o passar do tempo, naturalmente outros mineradores irão adicionar novos blocos ao final de suas próprias cópias do *blockchain* e irão propagá-los na rede. O sistema irá selecionar a cadeia com o maior número de blocos e a tornará definitiva. As cadeias restantes serão descartadas e as transações contidas em seus blocos voltarão para o conjunto de transações que estão aguardando para serem confirmadas. Devido à esse tipo de ocorrência, em várias criptomoedas é recomendado que os usuários aguardem até que mais blocos sejam adicionados após o bloco onde sua própria transação foi validada. Isso ajuda a garantir que a transação não seja desfeita.

A tecnologia de *blockchain* está se popularizando e sua aplicação como uma estrutura de armazenamento de dados vem sendo investigada em uma grande gama de sistemas em diversas áreas [IEEE 2017]. *Blockchains* podem (potencialmente) substituir platafor-

Figura 1.3: Bifurcação na cadeia de blocos.



mas digitais em diferentes setores, além da área de finanças. Porém, o investimento em pesquisa e desenvolvimento ainda é embrionário e necessário para acelerar a criação e comercialização de soluções baseadas nessa tecnologia.

1.2. Criptomoedas

A adoção de criptomoedas como forma de pagamento vem crescendo rapidamente devido aos benefícios que elas oferecem em relação às formas de pagamento tradicionais [medfar87 2018]. Um usuário pode efetuar um pagamento para um receptor em qualquer lugar do mundo a qualquer momento, sem a necessidade de uma instituição intermediária, o que leva a menores taxas de transações, maior controle, e mais privacidade. É importante notar que muitos dos benefícios oferecidos pelas criptomoedas em relação às instituições financeiras tradicionais, como descentralização e maior privacidade, são possibilitados pela utilização da tecnologia de *blockchains*. Entretanto, apesar dos benefícios oferecidos por esse tipo de moeda, existem desvantagens como a impossibilidade de reverter transações e de obter suporte caso ocorram erros no sistema. A volatilidade dos preços da moedas é outro fator negativo. No caso da Bitcoin, segundo estatísticas de mercado, o preço pode variar mais de 10% em poucas horas [Adkisson 2018].

1.2.1. Mineração

Para obter criptomoedas, um usuário pode realizar uma compra através de serviços especializados em vendas de criptomoedas, chamados de *cryptocurrency exchanges*. Algumas moedas, como a Bitcoin e a Monero, oferecem aos usuários a possibilidade de obtê-las diretamente através do sistema, participando de um processo comumente chamado de mineração. É importante notar que existem criptomoedas como a Ripple [Schwartz et al. 2014] e a IOTA [Popov 2014] que não possuem um sistema através do qual os usuários possam obter moedas ao criar blocos de transações, ou seja, não podem ser mineradas. Neste tutorial, iremos focar a discussão em processos de mineração similares aos que ocorrem em criptomoedas como a Bitcoin e a Monero.

O processo de mineração consiste em participar da criação de blocos válidos através do esquema de PoW, PoS ou outro similar, conforme determinado pelo respectivo

sistema. Esse processo é essencial para garantir o funcionamento do *blockchain* e da moeda digital, pois é através dele que as transações são confirmadas e adicionadas ao final da cadeia de blocos.

Mineradores são participantes da rede que utilizam sua capacidade computacional para tentar computar *digests* válidos para os blocos e adicioná-los ao final do *blockchain*. Quando um usuário efetua uma transação, seus dados são compartilhados na rede para que mineradores possam validá-la. Para cada transação efetuada, os usuários devem incluir uma taxa como pagamento para recompensar os mineradores. Transações que pagam taxas maiores são geralmente escolhidas primeiro e são incluídas em blocos mais rapidamente. Cada minerador escolhe as transações (disponíveis na rede) que irão fazer parte do bloco. Em seguida, inicia o processo de encontrar um *digest* que atenda às restrições do sistema.

Devido ao esforço computacional necessário para a criação de blocos válidos pelo esquema de PoW, é necessário um incentivo para que os participantes da rede, ou mineradores, emprestem o seu poder de processamento para ajudar no funcionamento do sistema. Esse incentivo é dado através de uma recompensa em criptomoedas para o participante da rede que conseguir validar primeiro um bloco de transações. Quando um minerador valida um bloco, sua informação é disseminada na rede para que os outros mineradores atualizem suas cópias do *blockchain* e escolham novas transações para validar.

A primeira transação de cada bloco, chamada de *coinbase transaction*, é um tipo especial de transação cuja finalidade é enviar o valor da recompensa para o usuário que efetuou a validação do bloco. O sistema de recompensas é geralmente desenvolvido de maneira que, com o passar do tempo, a recompensa por bloco diminua. Isto é necessário para controlar a quantidade de novas moedas criadas devido ao aumento no preço da moeda e outros fatores econômicos. Na Bitcoin, essa diminuição ocorre a cada 210.000 blocos minerados, quando a recompensa é reduzida pela metade.

A diminuição da recompensa estende a vida do sistema ao impedir que todo o suprimento de moedas seja emitido em um curto período de tempo, o que acabaria com a motivação para a criação de novos blocos válidos. A redução da recompensa também contribui para a valorização da moeda, que passa a valer mais quando a procura aumenta e a oferta diminui. Em um momento no futuro, a validação de novos blocos não irá gerar mais recompensas e o pagamento pela criação de blocos válidos será feito somente através das taxas de transações, pagas pelos usuários. Atualmente, em 2019, a quantia recompensada por bloco na Bitcoin é de 12,5 unidades da moeda, chamada de BTC.

A Tabela 1.1 mostra a mudança na recompensa por bloco válido criado ao longo do tempo. O período estimado para que ocorra a criação de 210.000 novos blocos e ocorra uma diminuição no valor da recompensa por bloco é de 4 anos. Na prática este período pode ser maior ou menor, apesar do aumento no número de mineradores ao longo do tempo. Esta oscilação ocorre porque o sistema eleva automaticamente a dificuldade de criação de novos blocos quando a velocidade da rede aumenta. Esta estratégia é empregada para manter o tempo entre a criação de novos blocos por volta de dez minutos. Isto também evita a emissão de muitas moedas novas em um curto período de tempo. Quando a dificuldade de criar blocos aumenta, o retorno pela criação de blocos diminui dado o esforço necessário, causando uma redução no número de mineradores. A redução

no número de mineradores diminui a capacidade de criação de blocos da rede, que ajusta a dificuldade de acordo. Essas variações causam mudanças no intervalo de tempo entre a diminuição da recompensa por bloco.

Tabela 1.1: Recompensa pela validação de blocos na Bitcoin.

Nº de Blocos	**Recompensa por bloco**	**Ano**
0	50 BTC	2009
210.000	25 BTC	2012
420.000	12,5 BTC	2016
630.000	6,25 BTC	2020 (estimado)

É comum que mineradores organizem-se em grupos que trabalham em conjunto para criarem blocos válidos, chamados de *mining pools*. Quando um usuário cria um bloco válido, a recompensa é dividida entre todos os participantes do grupo. Este método diminui o valor da recompensa individual de cada minerador, porém garante um fluxo mais estável de renda para todos. A participação em *mining pools* é interessante pelo fato de aumentar a probabilidade de retorno financeiro uma vez que é difícil um único usuário, sozinho, competir com os demais e as *mining pools*.

1.2.2. Hardware especializado para mineração

A mineração de blocos pode ser uma atividade lucrativa se um usuário conseguir vencer a corrida usando recursos computacionais a um custo menor do que a recompensa. Considerando a cotação atual da criptomoeda Bitcoin (43.520,00 reais em 05 de setembro de 2019), a recompensa de 12,5 BTC pela criação de um bloco é equivalente a 544.000,00 reais.

Com o objetivo de aumentar o lucro proveniente da mineração de criptomoedas, algumas empresas desenvolveram hardwares específicos para computar PoW da Bitcoin, chamados de circuitos integrados de aplicação específica, do inglês *Application Specific Integrated Circuit*s (ASICs). Esses equipamentos são projetados especificamente para computar *digests* de blocos em uma taxa muito superior àquela alcançável em hardware comum. Alguns dos ASICs mais poderosos, destinados ao sistema Bitcoin, como o *ANTMINER S9 Hydro* da fabricante Bitmain, possuem uma capacidade de processamento de até 18.000.000.000 (18 trilhões) de *hashes* por segundo, enquanto um processador Intel Core i7-3930k consegue computar apenas cerca de 98.000 *hashes* por segundo [Cointopper 2018].

O uso de ASICs permite que usuários monopolizem a criação de novos blocos, especialmente quando vários utilizadores desses dispositivos cooperam em *mining pools*. Algumas *Graphics Processing Unit*s (GPUs) também são utilizadas por mineradores para computar *hashes* por serem mais rápidas do que processadores comuns, porém, sua capacidade também é muito inferior aos *hardwares* especializados. A popularização de ASICs impede que usuários obtenham lucro através do processo de PoW do Bitcoin a menos que invistam na aquisição de equipamentos especializados [Jefferys 2018].

O sistema Monero utiliza um algoritmo resistente à ASIC, chamado de CryptoNight, no seu processo de PoW. Este algoritmo é usado para tornar o processo de mineração mais igualitário do que em outras criptomoedas. O algoritmo CryptoNight pertence a uma classe de funções denominada *memory-bound*. Nessa classe de função, o tempo necessário para resolver um problema computacional depende principalmente da quantidade e da velocidade da memória disponível para armazenar os dados utilizados durante a sua resolução. A utilização do algoritmo CryptoNight combate o uso de ASICs, que tornaram-se praticamente essenciais para usuários que desejam participar do processo de PoW da Bitcoin.

1.2.3. Mineração através de navegadores

A idéia de inserir códigos de mineração de criptomoedas em páginas da *Web* surgiu logo após o lançamento da Bitcoin [Eskandari et al. 2018]. Pouco tempo após a proliferação da idéia, várias aplicações de mineração desenvolvidas com a linguagem de programação JavaScript tornaram-se populares como: JSMiner[4], MineCrunch[5] e Tidbit[6]. O processo de mineração utilizando navegadores foi divulgado como uma alternativa à exibição de propagandas para a monetização de conteúdos em páginas da *Web*. Ao visitar uma página que contém uma aplicação de mineração, os recursos computacionais do usuário são utilizados para computar *digests* de blocos de transações, uma etapa essencial na criação de blocos e obtenção de recompensas em sistemas de criptomoedas.

Algumas páginas *Web* defendem o uso benigno de mineradores baseados em código JavaScript para a geração de lucro e manutenção de sua infraestrutura [Saad et al. 2018], solicitando que os usuários emprestem seu poder computacional em troca de acesso ao conteúdo da página. Entretanto, o uso dessas aplicações por usuários maliciosos difundiu-se rapidamente devido à facilidade de execução de ataques, necessitando apenas que as vítimas possuam uma conexão com a Internet e visitem uma página da *Web* infectada com o código do minerador desenvolvido em JavaScript. Após a difusão inicial da idéia e o surgimento de diversas aplicações de mineração via navegadores, desenvolvedores e páginas da *Web* que compactuavam com o uso desse tipo de aplicações passaram a enfrentar problemas judiciais devido ao uso não autorizado dos recursos computacionais dos usuários [Blattberg 2014].

No ano de 2017, anos após a primeira onda de aplicações de mineração codificadas em JavaScript, as atividades de mineração em navegadores aumentaram expressivamente devido ao lançamento de um novo serviço de mineração de criptomoedas em plataformas *Web*, chamado de Coinhive [Rauchberger et al. 2018]. O código de mineração do serviço Coinhive chegou a estar presente em cerca de 32.000 *websites* distintos em 2017 [Krebs 2018]. Desta vez, o foco da mineração deixou de ser a Bitcoin e voltou-se para a criptomoeda Monero. Um dos principais fatores que contribuiram para a escolha da Monero foi o seu algoritmo de PoW, CryptoNight, projetado para permitir que computadores com processadores de propósito geral sejam adequados para participar do processo de criação de blocos. Ao permitir que os processadores presentes em computadores comuns sejam capazes de calcular *hashes* de blocos de maneira mais eficiente

[4] https://github.com/jwhitehorn/jsMiner
[5] https://www.technicpack.net/modpack/minecrunch.814457/changelog
[6] https://en.wikipedia.org/wiki/Tidbit

do que *hardwares* especializados, o algoritmo CryptoNight torna viável a mineração de Monero através de códigos embutidos em páginas *Web*.

O serviço Coinhive despertou novamente o interesse de atacantes. A mineração de criptomoedas através de navegadores sem o consentimento dos usuários (*in-browser cryptojacking*) tornou-se novamente uma forma de ataque proeminente. Ataques de *cryptojacking* consistem na utilização dos recursos computacionais de uma ou mais vítimas, sem o seu consentimento, para a realização do processamento necessário à criação de blocos válidos em criptomoedas. No ano de 2017, o aumento na detecção de ataques de *in-browser cryptojacking* foi de 8.500% [Mathur 2018]. Em junho do ano de 2018, a plataforma Coinhive foi responsável por 1,18% dos blocos minerados no sistema Monero [Rüth et al. 2018].

Estima-se que 10 milhões de usuários sejam afetados por ataques de *in-browser cryptojacking* a cada mês [Hong et al. 2018]. Páginas da *Web* nas quais os usuários permanecem por longos períodos de tempo como plataformas de *streaming* de vídeo, são alvos ideais para a injeção de scripts de *cryptojacking*, visto que o lucro obtido através de mineração é proporcional ao tempo que os usuários permanecem em uma página infectada.

A execução de um ataque de *in-browser cryptojacking* geralmente depende de dois componentes distintos: um código controlador e um código de mineração. A Figura 1.4 ilustra o processo de comunicação necessário entre a máquina do usuário e os servidores do atacante para iniciar ao processo de mineração. A primeira etapa da comunicação ocorre quando um usuário visita uma página que executa um código de mineração. Ao comunicar-se pela primeira vez com a página, o navegador do usuário recebe e executa um código Javascript que inspeciona os recursos computacionais disponíveis em sua máquina. O script também verifica se o navegador da vítima suporta a execução de código WebAssembly (Wasm).

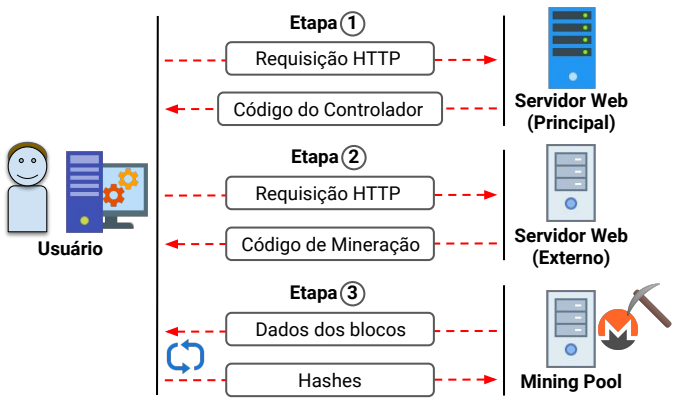
O formato de instruções Wasm é otimizado para permitir a execução de código em navegadores com um desempenho próximo ao de uma aplicação sendo executada nativamente na máquina[7]. Na próxima etapa, o script comunica-se com um servidor externo definido pelo atacante, isto é, um servidor diferente do qual o usuário acessou. Se o navegador suportar a execução de código Wasm, o script do controlador faz o download de um código Wasm que é compilado na máquina da vítima. Caso contrário, é realizado o download de instruções `asm.js`, um tipo de código JavaScript focado em desempenho.

Na última etapa, o código de mineração cria *threads* (processos leves) na máquina da vítima de acordo com a quantidade de recursos disponíveis, cria uma conexão com um serviço de *mining pool* e requisita tarefas de processamento. Ao terminar as tarefas recebidas, a máquina envia para o servidor da *mining pool* os *digests* computados e repete a última etapa até o encerramento da conexão.

Embora sejam amplamente utilizadas, estima-se que plataformas de mineração como Coinhive não gerem um retorno monetário tão expressivo quanto a exibição de propagandas em páginas da *Web* [Saad et al. 2018]. A menos que as páginas utilizadoras de mineração em navegadores atraiam muitos usuários e ofereçam conteúdos que os man-

[7] https://webassembly.org/

Figura 1.4: Diagrama ilustrando as etapas um ataque de *in-browser cryptojacking*.



tenham no *site* por longos períodos de tempo, a utilização desse tipo de serviço não é recomendado.

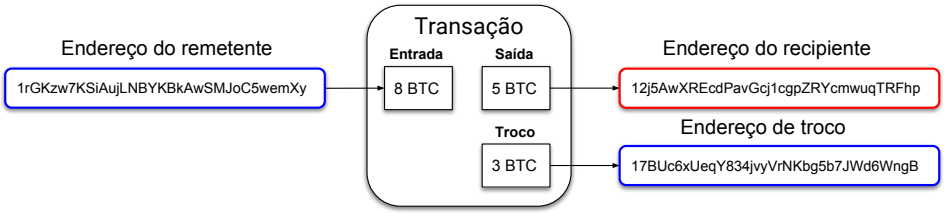
1.3. A criptomoeda Bitcoin

Introduzida através de um *white paper* em uma lista de correio eletrônico sobre criptografia em 2008 e lançada em 2009, Bitcoin foi a primeira criptomoeda de sucesso e utilizada em larga escala [Nakamoto 2008]. Anos após seu lançamento, ainda permanece sendo a mais utilizada, possuindo um valor aproximado de mercado de US$ 188 bilhões[8] em 05 de setembro de 2019. A Bitcoin baseia-se em material de pesquisas anteriores, como o esquema de PoW para controlar a criação de blocos válidos no *blockchain*, esquemas de assinaturas digitais para garantir que os usuários que utilizam moedas realmente as possuem e técnicas de *timestamping* que marcam a data e a hora da realização das operações. A principal contribuição da Bitcoin foi eliminar a necessidade de uma autoridade central que regule a emissão de moedas e a confirmação de transações. Isso foi possível pela forma descentralizada pela qual o sistema funciona, utilizando uma rede ponto-a-ponto onde usuários participam do processo de validação e verificação da autenticidade das transações. A descentralização também é fruto da maneira como os dados estão armazenados. Todas as transações ocorridas estão armazenadas em um *blockchain* público, isto é, cada participante da rede pode optar por obter uma cópia desse registro. Usuários que não desejam realizar o download dos dados do *blockchain* da criptomoeda podem acessá-los através de clientes leves, do inglês *Light-Clients*. Clientes leves são programas que permitem aos usuários acessar dados através de uma conexão com um nó remoto confiável que mantém uma cópia do *blockchain*.

Para enviar e receber transações em Bitcoin, um usuário necessita de um par de chaves criptográficas composto por uma chave pública e uma chave privada. No caso da Bitcoin, a chave pública é utilizada como endereço de envio e recebimento de paga-

[8] https://coinmarketcap.com/

Figura 1.5: Transação de Bitcoins contendo uma única entrada.



mentos. Um usuário pode divulgar a sua chave pública e outras pessoas podem enviar Bitcoins para esse endereço. A chave privada é utilizada para comprovar que um usuário é dono da chave pública que a acompanha, podendo assim utilizar os fundos recebidos. A chave privada não deve ser compartilhada e deve ser armazenada em segurança para que ninguém além do proprietário possa acessá-la. O endereço de um usuário é derivado de sua chave pública, ao aplicar o algoritmo de *hashing* SHA-256 e depois o algoritmo RIPEMD-160, adicionar números para controle de erro e controle de versões e, por fim, codificá-lo em BASE58 [Tschorsch and Scheuermann 2016]. O processo de derivação da chave é realizado para fornecer segurança adicional, ajudando a ocultar a verdadeira chave pública. Os usuários também podem optar pela utilização de sua chave pública original como endereço.

Em uma transação, existem endereços de entrada e endereços de saída. A Figura 1.5 ilustra o procedimento de uma transação de Bitcoins contendo uma única entrada. O endereço do remetente é a chave pública que corresponde ao endereço da carteira de Bitcoins do emissor do pagamento. O endereço do recipiente é o endereço da carteira do receptor do pagamento. O endereço de troco deve ser definido pelo emissor do pagamento para que ele receba o troco da operação, caso a chave de entrada utilizada exceda o valor do pagamento.

Só podem ser utilizadas como entradas de transações as chaves que foram geradas como saída em uma transação anterior. Endereços de saída de transações são chaves recebidas pelos usuários que recebem as criptomoedas. O saldo de um usuário da Bitcoin consiste na soma dos valores de todas as saídas de transações que ele já recebeu e ainda não utilizou. Antes de serem utilizadas, as chaves que contêm criptomoedas recebem a denominação de "saída de transação não-utilizada", do inglês *Unspent Transaction Output* (UTXO). Sempre que uma chave de entrada é utilizada em uma transação, todo o seu conteúdo em Bitcoins deve ser gasto, ou seja, não é possível usar somente parte da quantia armazenada em um endereço. Após uma UTXO ser utilizada, seu estado muda para "chave de saída utilizada", do inglês *Spent Transaction Output* (STXO), para indicar que a quantia armazenada nesta chave já foi gasta. Para permitir que os remetentes mantenham o dinheiro que sobra do pagamento, existe a idéia de troco, onde o pagamento é feito para o recipiente e o restante é enviado para um endereço de escolha do remetente. O endereço de troco pode ser o mesmo endereço utilizado como entrada, mas isso é de-

Figura 1.6: Estrutura de um bloco do *blockchain* do sistema Bitcoin.

sencorajado porque quanto mais um endereço é utilizado, mais fácil torna-se o processo de rastrear informações do usuário. É recomendado que os usuários utilizem uma carteira de Bitcoins, um tipo de programa que auxilia no gerenciamento das chaves e endereços ao criar automaticamente novos endereços para o recebimento de troco. Uma carteira é capaz de gerenciar diferentes endereços de um mesmo usuário.

Um único bloco é capaz de armazenar milhares de transações, desde que os dados de todas elas somados não ultrapassem o tamanho de 1 MB. A Figura 1.6 resume os dados contidos um bloco do sistema Bitcoin. Os blocos criados para armazenar as transações no *blockchain* possuem os seguintes campos:

(c_1) *Número mágico*: Valor utilizado para identificar o tipo de estrutura contida nos dados. Neste caso, um bloco. Este valor é específico do protocolo Bitcoin.

(c_2) *Tamanho do bloco*: Especifica o tamanho em *bytes* dos dados contidos no bloco.

(c_3) *Cabeçalho do bloco*: Contém dados que identificam o bloco atual.

(c_4) *Versão*: Especifica a versão do sistema no momento da criação do bloco.

(c_5) *Timestamp*: Representa o momento no tempo em que o bloco foi criado.

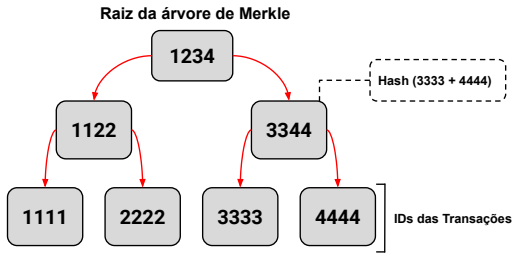
(c_6) *Raiz Merkle*: Um tipo de *hash* utilizado para verificar a validade das transações contidas nos blocos sem a necessidade de verificar todas as informações das transações. Para realizar a verificação é necessário o cabeçalho dos blocos e uma estrutura chamada de Árvore de Merkle.

(c_7) *Nonce*: Campo cujo valor deve ser modificado até que o *digest* resultante do bloco atenda as exigências do sistema.

(c_8) *Dificuldade*: Especifica o número de *bits* 0 necessários à esquerda do *digest* do bloco para que ele atenda às exigências do sistema.

(c_9) *Número de transações*: Contém o número de transações presentes no bloco atual.

Figura 1.7: Representação de uma Árvore de Merkle.



(c_{10}) *Transações*: Contém os dados de cada uma das transações contidas no bloco.

É interessante ressaltar a importância do campo que armazena a raiz da Árvore de Merkle. A Figura 1.7 ilustra a estrutura de uma Árvore de Merkle. Os valores contidos nos nós da árvore foram simplificados para fins didáticos. Em uma estrutura real, as informações armazenadas dentro dos nós de uma Árvore de Merkle são os *digests* gerados pela função SHA-256. A raiz de Merkle funciona como um identificador para as transações que estão contidas em um bloco.

Imagine que para cada bloco criado seja computado um resumo (ou *digest*) dos identificadores de todas as transações nele contidas. Para verificar se todas as transações contidas no bloco foram incluídas no *digest* (para verificar a integridade dos dados), seria necessário reunir os identificadores de todas as transações e computar o *digest* novamente. Em uma Árvore de Merkle, cada nó da estrutura armazena o *digest* dos dados contidos em seus nós filhos. Os nós folha da árvore armazenam os dados originais, no caso da Bitcoin, os identificadores das transações contidas no bloco. Com essa estrutura é possível verificar a integridade das transações presentes em um bloco através dos *digests* que resumem os dados, acelerando o processo.

Considerando a estrutura apresentada na Figura 1.7, suponha que um usuário foi informado de que recebeu Bitcoins através da transação com o identificador 4444, em um bloco cuja raiz de Merkle é 1234. O receptor deseja verificar se esta transação está de fato presente no bloco. O usuário já conhece o valor da raiz da árvore e do identificador de sua transação. Para realizar a verificação, serão necessários também os valores 3333 e 1122. Primeiro, é computado o *digest* dos valores 3333 e 4444 e é gerado o valor 3344, que resume os dados das folhas do lado direito da árvore. O valor que resume os dados das folhas do lado esquerdo, 1122, já é conhecido. Na sequência é gerado um *digest* a partir dos valores que resumem os dois lados da árvore, 1122 e 3344, gerando a raiz da Árvore de Merkle, que é igual a 1234 e resume ambos os lados da árvore. Se a raiz de Merkle resultante é igual ao valor contido dentro do bloco no *blockchain*, isto significa que a transação 4444 está contida nas transações do bloco examinado.

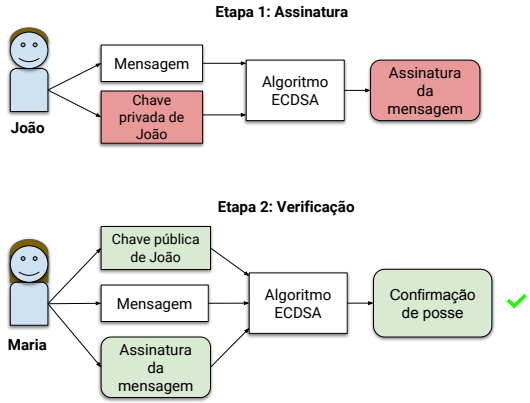
Em blocos com poucas transações, a diferença entre o número de dados necessários para realizar a verificação e o número de dados no bloco é pequena. Porém, conforme o número de transações aumenta para centenas ou milhares, o número de operações com uma Árvore de Merkle torna-se muito menor do que o número de transações ne-

cessárias para verificar o *digest*, tornando evidente o benefício da utilização de Árvores de Merkle.

Como o *blockchain* da Bitcoin é um registro transparente, o histórico de transações de todos os usuários está disponível abertamente. Através da análise dos endereços e do fluxo de transações é possível efetuar ataques que correlacionam os pseudônimos com as identidades dos usuários [Meiklejohn et al. 2013]. Para reduzir o impacto de ataques que efetuam a análise das transações, é sugerida a utilização de uma nova chave e endereço para cada transação [Nakamoto 2008]. A quantidade de Bitcoins associada a cada usuário não é armazenada explicitamente nos registros. O saldo de um endereço pode ser verificado ao checar todo o seu histórico de transações, calculando quantas Bitcoins foram recebidas e quantas foram enviadas a partir do endereço. Por isso, sempre que um usuário instala pela primeira vez uma carteira de Bitcoins em seu sistema, é necessário que ele verifique todas as transações já ocorridas. A verificação é realizada para checar se os pagadores possuem de fato os valores sendo gastos.

Para garantir a segurança das transações na Bitcoin, um sistema de assinaturas baseado no algoritmo ECDSA é utilizado. A Figura 1.8 ilustra o processo de criação de uma assinatura digital. João precisa provar que possui um endereço para enviar dinheiro através dele. Para isso, ele cria uma mensagem contendo os dados da transação que deseja realizar. O segundo passo é criar uma assinatura digital, que servirá para provar que João é o dono do endereço do qual as moedas estão sendo enviadas. Utilizando a sua chave privada, João gera a assinatura digital da mensagem e a envia para a rede juntamente com a mensagem e sua chave pública.

Figura 1.8: Processo de assinatura de uma transação.



Para verificar a validade da transação, Maria realiza uma operação matemática utilizando a chave pública e assinatura enviadas junto com a mensagem. O resultado da operação irá confirmar se a chave pública recebida corresponde à chave privada utilizada na geração da assinatura digital, sem revelar qualquer informação sobre a chave privada. Maria saberá que a chave pública enviada junto com a mensagem, que é o mesmo

endereço de onde estão sendo enviadas moedas, pertence à pessoa que tem chave privada correspondente. João é então identificado como o dono do endereço.

Após a criação de uma transação, o remetente dissemina na rede uma mensagem avisando que possui uma nova transação. Os participantes interessados nos dados enviam um pedido explícito ao remetente. Os mineradores acumulam transações e as organizam em blocos antes de iniciarem o processo de tentativa de criação de um bloco válido. A dificuldade de criação dos blocos é regulada pelo sistema e é alterada com base no tempo que foi necessário para a criação dos últimos 2.016 blocos. O ajuste é realizado com a intenção de manter o tempo necessário para adicionar um bloco a cada dez minutos, para que o tempo de confirmação das transações seja razoável. Levando em consideração o tempo ideal de criação de um bloco, 10 minutos, 2.016 blocos devem ser criados em exatamente duas semanas. Se o tempo necessário para a criação dos últimos 2.016 blocos exceder duas semanas, a dificuldade da criação dos blocos é reduzida e se o tempo for inferior a duas semanas, a dificuldade é elevada. O sistema ajusta a dificuldade da criação de blocos válidos de acordo com a Equação 1.

$$D = D_{anterior} \times \frac{T}{2016 \times 10min} \tag{1}$$

onde:

D = Dificuldade da resolução do problema de criação de um bloco válido.

T = Tempo ocorrido desde a última mudança de dificuldade em minutos.

Ao efetuar a criação de um bloco válido, o minerador dissemina a informação do bloco para que os outros participantes saibam que as transações contidas naquele bloco foram confirmadas por ele. Uma aplicação de carteira Bitcoin realiza uma varredura na cadeia de blocos para verificar as transações destinadas ao seu endereço público e saber quantas moedas o usuário possui. Devido à transparência das informações contidas no *blockchain*, qualquer um com acesso aos dados é capaz de descobrir o saldo de um endereço qualquer, diminuindo a privacidade dos usuários.

Apesar de apresentar alguns problemas de privacidade e de possuir uma capacidade limitada de processar transações devido ao esquema de PoW, a Bitcoin continua sendo amplamente utilizada. Seu sucesso contribui para a criação de novas criptomoedas que buscam solucionar problemas existentes nos sistemas atuais, como a demora das confirmações e a rastreabilidade das transações.

1.4. UniBlock

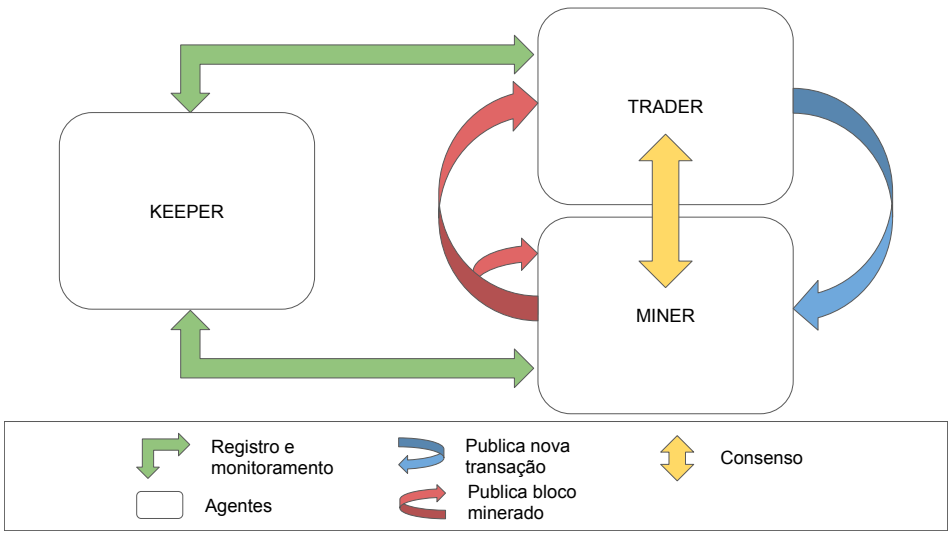
Para demonstrar na prática o funcionamento de uma estrutura de *blockchain*, de maneira didática, foi desenvolvido um sistema denominado UniBlock. Este sistema permite aos seus usuários executar as suas próprias instâncias de entidades cruciais de uma rede *blockchain*, como os mineradores (*miners*) e negociantes (*traders*). Estas entidades participam do processo de criação de transações e mineração de blocos, conforme ilustrado na Figura 1.9.

O desenvolvimento do UniBlock foi baseado na estrutura de *blockchain* do tutorial "*Learn Blockchains by Building One*"[9]. O tutorial destina-se aos usuários iniciantes, que querem conhecer os princípios básicos da implementação de uma *blockchain*. Entretanto, a implementação da *blockchain* não apresenta uma separação clara entre mineradores e negociantes, afetando o entendimento e a compreensão do sistema. Suprir esta deficiência do código apresentado no tutorial foi um dos principais motivadores para a criação do UniBlock[10].

1.4.1. Arquitetura e Funcionamento do UniBlock

A arquitetura do UniBlock pode ser visualizada na Figura 1.9. Como pode ser observado, há três agentes essenciais, o *Keeper*, o *Trader* e o *Miner*. O *Keeper* é o serviço de coordenação das entidades distribuídas da rede da *blockchain*, cuja finalidade e funcionalidades, apesar de simplificadas, são similares as do Apache ZooKeeper[11]. Os *Traders* são responsáveis por gerar as transações que irão alimentar a cadeia de blocos. Por fim, os *Miners* são responsável por minerar os blocos alimentados preenchidos com transações enviadas pelos *Traders*. Vale ressaltar que há somente um único *Keeper* ativo na rede, enquanto que o número de *Miners* e *Traders* pode variar de acordo com o cenário de testes que o usuário pretende criar.

Figura 1.9: Representação da arquitetura do UniBlock.



A rede do UniBlock é iniciada obrigatoriamente com um *Keeper*, que irá manter atualizada a lista de entidades ativas na rede. Em seguida, *Miners* e *Traders* podem ser criados. Cada novo agente irá registrar-se ao *Keeper*. Além disso, cada agente da rede

[9] https://github.com/dvf/blockchain
[10] https://github.com/homdreen/UniBlock
[11] https://zookeeper.apache.org/

deverá enviar, periodicamente, um *heartbeat* ao *Keeper*, informando que ainda está ativo na rede. Caso o *heartbeat* não seja recebido, o *Keeper* remove o agente da rede. Neste caso, o *Keeper* irá informar todos os agentes ativos sobre a remoção do agente cujo *heartbeat* não foi recebido. Na Figura 1.9, este processo é representado pelas setas verdes de registro e monitoramento.

Os dados utilizados nas transações são alimentados nos *Traders*. Assim que uma nova transação é criada, o *Trader* publica a nova transação aos *Miners* ativos na rede (seta azul da figura). Os *Miners* agrupam as transações e mineram os blocos que serão publicados na rede (setas vermelhas da figura). Finalmente, para ser inserido definitivamente na *chain*, o novo bloco precisa ser aprovado pela maioria (i.e., mais de 50% do agentes ativos), utilizando um protocolo de consenso entre os *Traders* e *Miners* da rede (seta amarela na figura).

1.4.2. Utilização Prática do UniBlock

Todos os agentes do UniBlock são inicializados a partir do mesmo código Python (`main.py`). Primeiro, é necessário inicializar um *Keeper*, passando como parâmetro a opção `--keeper`, como ilustrado a seguir. É necessário passar como parâmetro também o IP e a porta onde o *Keeper* estará aguardando as conexões dos *Traders* e *Miners* da rede.

```
$ python3 main.py -ki <keeperIP> -kp <keeperPorta> --keeper
```

Para adicionar um novo minerador no sistema, é necessário fornecer o IP e a porta do *Keeper*, além de incluir o parâmetro `--miner`, que identifica o agente como sendo um *Miner*. O minerador possui uma interface que permite acessar informações em tempo real, como os usuários ativos no sistema, a carteira atual e os dados da cadeia de blocos. Os comandos da interface estão disponíveis ao usuário através do comando `help`.

```
$ python3 main.py -ki <keeperIP> -kp <keeperPorta> --miner
```

A inicialização de um *trader* é similar à de um minerador, incluindo o IP e porta do *Keeper* e o parâmetro `--trader`, que o identificará como sendo um *Trader*. Assim como o negociador, o minerador oferece um conjunto de comandos, que podem ser visualizados através do comando `help`, para acessar informações em tempo real. No *Trader*, o usuário do sistema pode listar usuários ativos, criar uma nova transação ou listar os dados do *blockchain*.

```
$ python3 main.py -ki <keeperIP> -kp <keeperPorta> --trader
```

1.5. Considerações Finais

Este minicurso apresentou uma introdução às tecnologias de *blockchain* e criptomoedas, desde a teoria necessária para entender essas tecnologias até as estruturas e métodos utilizados. As discussões foram acompanhadas de ilustrações e exemplos com o objetivo de auxiliar a compreensão do leitor. Temas como os métodos de segurança e consenso foram

abordados, tendo em vista que estes garantem o correto funcionamento da *blockchain*, um sistema descentralizado, sem a necessidade de um intermediador para que as transações aconteçam. Também foram discutidos os detalhes de funcionamento da criptomoeda Bitcoin, que é a moeda digital mais utilizada atualmente.

No minicurso foi apresentado também o UniBlock, um sistema de *blockchain* com fins didáticos. De uma forma simples e bem estruturada, o UniBlock apresenta os conceitos, a implementação e a prática com uma rede de negociadores (*Traders*) e mineradores (*Miners*). Com o UniBlock, os usuários podem rapidamente criar, operacionalizar e acompanhar os detalhes de funcionamento de uma *blockchain*.

Referências

[Abbas and Sung-Bong 2019] Abbas, Q. E. and Sung-Bong, J. (2019). A Survey of Blockchain and Its Applications. In *ICAIIC*, pages 001–003.

[Adkisson 2018] Adkisson, J. (2018). Why bitcoin is so volatile. http://tiny.cc/8fp35y.

[Al-Jaroodi and Mohamed 2019] Al-Jaroodi, J. and Mohamed, N. (2019). Blockchain in Industries: A Survey. *IEEE Access*, 7:36500–36515.

[Blattberg 2014] Blattberg, E. (2014). New Jersey slaps MIT Bitcoin hackers with subpoena — and they're fighting back. https://tinyurl.com/y3owapf4.

[Chen et al. 2018] Chen, W., Xu, Z., Shi, S., Zhao, Y., and Zhao, J. (2018). A survey of blockchain applications in different domains. In *Proceedings of the ICBTA*, pages 17–21, New York, NY, USA. ACM. http://doi.acm.org/10.1145/3301403.3301407.

[Cointopper 2018] Cointopper (2018). Difference between asic, gpu and cpu mining. http://tiny.cc/ljp35y.

[Dai et al. 2019] Dai, H., Zheng, Z., and Zhang, Y. (2019). Blockchain for internet of things: A survey. *CoRR*, abs/1906.00245.

[Eskandari et al. 2018] Eskandari, S., Leoutsarakos, A., Mursch, T., and Clark, J. (2018). A first look at browser-based cryptojacking. *arXiv preprint arXiv:1803.02887*. https://arxiv.org/abs/1803.02887.

[Hong et al. 2018] Hong, G., Yang, Z., Yang, S., Zhang, L., Nan, Y., Zhang, Z., Yang, M., Zhang, Y., Qian, Z., and Duan, H. (2018). How you get shot in the back: A systematical study about cryptojacking in the real world. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1701–1713. ACM. https://doi.org/10.1145/3243734.3243840.

[Hoy 2017] Hoy, M. B. (2017). An introduction to the blockchain and its implications for libraries and medicine. *Medical reference services quarterly*, 36(3):273–279.

[IEEE 2017] IEEE (2017). Special Report on Blockchain World. *IEEE Spectrum*, 10. http://bit.do/spectrum-blockchain.

[Jefferys 2018] Jefferys, K. (2018). The Problem With ASICs. http://tiny.cc/kqu35y.

[Krebs 2018] Krebs, B. (2018). Who and what is coinhive? https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive/.

[Marinoff 2018] Marinoff, N. (2018). South korea is trialing blockchain voting — here's what that means. http://tiny.cc/swp35y.

[Mathur 2018] Mathur, N. (2018). Cybersecurity: Cryptojacking attacks exploded by 8,500% in 2017, says report. https://tinyurl.com/y84alobt.

[medfar87 2018] medfar87 (2018). Cryptocurrency Growth & Adoption Statistics. http://tiny.cc/oxp35y.

[Meiklejohn et al. 2013] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M., and Savage, S. (2013). A fistful of bitcoins: characterizing payments among men with no names. In *ACM IMC*, pages 127–140. ACM.

[Min et al. 2019] Min, T., Wang, H., Guo, Y., and Cai, W. (2019). Blockchain Games: A Survey. *CoRR*, abs/1906.05558.

[Nakamoto 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf.

[NIST 2018] NIST (2018). Hash functions. https://csrc.nist.gov/projects/hash-functions.

[Olnes et al. 2017] Olnes, S., Ubacht, J., and Janssen, M. (2017). Blockchain in government: Benefits and implications of distributed ledger technology for information sharing. *Government Information Quarterly*, 34(3):355 – 364.

[Popov 2014] Popov, S. (2014). The tangle. http://www.tangleblog.com/wp-content/uploads/2016/11/IOTA_Whitepaper.pdf.

[Rauchberger et al. 2018] Rauchberger, J., Schrittwieser, S., Dam, T., Luh, R., Buhov, D., Pötzelsberger, G., and Kim, H. (2018). The other side of the coin: A framework for detecting and analyzing web-based cryptocurrency mining campaigns. In *Proceedings of the 13th Int. Conf. on Availability, Reliability and Security*, page 18. ACM.

[Rüth et al. 2018] Rüth, J., Zimmermann, T., Wolsing, K., and Hohlfeld, O. (2018). Digging into Browser-based Crypto Mining. In *ACM IMC*, pages 70–76. ACM.

[Saad et al. 2018] Saad, M., Khormali, A., and Mohaisen, A. (2018). End-to-End Analysis of In-Browser Cryptojacking. *arXiv preprint arXiv:1809.02152*. https://arxiv.org/abs/1809.02152.

[Schwartz et al. 2014] Schwartz, D., Youngs, N., Britto, A., et al. (2014). The Ripple protocol consensus algorithm. https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[Tschorsch and Scheuermann 2016] Tschorsch, F. and Scheuermann, B. (2016). Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials*, 18(3):2084–2123.

[Xie et al. 2019] Xie, J., Tang, H., Huang, T., Yu, F. R., Xie, R., Liu, J., and Liu, Y. (2019). A Survey of Blockchain Technology Applied to Smart Cities: Research Issues and Challenges. *IEEE Communications Surveys Tutorials*, 21(3):2794–2830.

[Yang et al. 2019] Yang, R., Yu, F. R., Si, P., Yang, Z., and Zhang, Y. (2019). Integrated Blockchain and Edge Computing Systems: A Survey, Some Research Issues and Challenges. *IEEE Communications Surveys Tutorials*, 21(2):1508–1532.

[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

Introdução à Propriedades Básicas e Avançadas de Segurança da Informação

Diego Kreutz, Sabrina Carlé Winckler, Rodrigo de Oliveira Barbosa, João Otávio Chervinski, Tadeu Jenuário (Unipampa)

A segurança e a privacidade da informação continuam sendo os principais desafios de desenvolvedores de sistemas, empresas e outras instituições. Casos recentes e cada vez mais frequentes, como o divulgado pelo New York Times [Times 2018], alertam para a criticidade da situação atual. No caso, a empresa Check Point Software Technologies, especializada em ciber segurança, encontrou uma maneira de alterar mensagens do aplicativo WhatsApp[1], que pertence ao Facebook e é utilizado por cerca de 1.5 bilhões de pessoas. De acordo com a Check Point Software Technologies, ao criar uma versão modificada do aplicativo, golpistas podem mudar uma "citação" (função que permite usuários em uma conversa visualizar mensagens antigas e responder às mesmas) passando a impressão que uma pessoa enviou uma mensagem que, na realidade, ela não enviou. Isto pode ser caracterizado em segurança como personificação, ou seja, um atacante se passando por outra pessoa para conseguir obter informações privadas do alvo do ataque.

Outro exemplo, ocorrido em 2018, é o maior vazamento de dados já registrado, que comprometeu mais de 1,1 bilhões de registros de dados pessoais de cidadãos da Índia [Leskin 2018, Machado et al. 2019]. É interessante observar que nos anos anteriores, de 2014 a 2017, não foi diferente [Machado et al. 2019]. Os principais incidentes de segurança e vazamentos de dados sensíveis foram na ordem de centenas de milhões de dados de usuários. Vale ressaltar também que estudos recentes demonstram um cenário preocupante no ecosistema HTTPS de países como o Brasil, incluindo certificados com problemas e o suporte a versões do TLS/SSL vulneráveis a ataques conhecidos [Escarrone et al. 2019]. Esses dados reais, atuais, ilustram a importância, cada vez maior, de propriedades básicas e avançadas de segurança para garantir tanto a segurança quanto a privacidade dos dados dos usuários.

As propriedades mais fundamentais de segurança são confidencialidade, integridade e autenticidade (CIA[2]). A confidencialidade dos dados é normalmente garan-

[1] https://www.whatsapp.com

[2] Neste minicurso utilizaremos CIA para denominar Confidencialidade, Integridade e Autenticidade. No inglês, utiliza-se CIA para denominar a tríade de *Confidentiality, Integrity, and Avaliability*.

tida por algoritmos de criptografia simétrica (e.g., AES[3] e 3DES[4]) ou assimétrica (e.g., RSA [Rivest et al. 1978]). O dado original é utilizado como uma das entradas desses algoritmos. A segunda entrada é uma chave secreta (criptografia simétrica) ou uma chave pública (criptografia assimétrica). A partir de um conjunto de operações matemáticas sobre os dados e as respectivas chaves de entrada, são gerados conjuntos de *bits* sem significado por si só, isto é, uma verdadeira "sopa de letrinhas". Quando utilizado um algoritmo de criptografia simétrica, o destinatário dos dados precisa aplicar uma função do mesmo algoritmo (e.g., AES), com a mesma chave secreta utilizada pelo remetente para visualizar os dados originais da mensagem. No caso da utilização de um algoritmo assimétrico, para a codificação é tipicamente utilizada a chave pública, enquanto que a chave privada é utilizada para a decodificação. Desta forma, a chave pública pode ser divulgada publicamente, ou seja, sem qualquer tipo de restrição. Já a informação poderá ser decodificada apenas com a chave privada correspondente, que é conhecida apenas pelo usuário proprietário do respectivo par de chaves pública/privada.

Nos exemplos apresentados a seguir, utilizamos Alice e Bob como as duas entidades (e.g., duas pessoas ou dois programas modelo cliente/servidor) que estão compartilhando dados através de mensagens. Na comunicação entre Alice e Bob, a confidencialidade garante apenas que os dados originais (decifrados) não são passíveis de leitura por um agente malicioso Eve[5], que tem acesso apenas aos dados cifrados. Porém, a confidencialidade não garante a integridade nem a autenticidade dos dados. Na prática isto significa que Eve pode modificar os dados cifrados que estão sendo enviados da Alice para o Bob e vice-versa. A priori, Bob irá receber os dados cifrados e não tem como saber se eles foram modificados em trânsito. Para verificar a integridade, Alice pode enviar também uma súmula (mais conhecida como *digest*) dos dados cifrados, que pode ser gerada utilizando uma função de *hash* criptográfica segura (e.g., SHA-256, SHA-512, SHA3-256, SHA3-512[6] [Homsirikamol et al. 2012]). Entretanto, Eve ainda pode alterar os dados cifrados, re-gerar a súmula e enviar todos os dados alterados para Bob.

A autenticidade é propriedade de segurança que falta para Bob conseguir verificar a integridade e origem dos dados. Assumindo que Alice e Bob compartilham uma chave secreta K, Alice cifra os dados utilizando esta chave e gera um código de autenticação de mensagem, mais conhecido como HMAC (*Hash-based Message Authentication Code*)[7] [Krawczyk et al. 1997]. O HMAC dos dados enviados pela Alice permite ao Bob verificar tanto a integridade quanto a autenticidade dos dados recebidos. O HMAC é uma espécie de assinatura da súmula dos dados. Como Bob consegue verificar que a súmula veio de Alice através da chave K, que apenas ambos conhecem, ele consegue também confirmar a integridade dos dados cifrados, ou seja, consegue saber se houve ou não alteração dos dados no meio do caminho. Portanto, combinando confidencialidade, integridade e autenticidade, Alice e Bob conseguem, finalmente, estabelecer comunicações

[3] https://csrc.nist.gov/projects/block-cipher-techniques

[4] https://csrc.nist.gov/news/2017/update-to-current-use-and-deprecation-of-tdea

[5] Utilizaremos Eve para denominar o agente malicioso que tenta interceptar e comprometer as comunicações entre duas entidades como Alice e Bob.

[6] https://www.nist.gov/publications/secure-hash-standard

[7] Apesar de MAC (*Message Authentication Code*) ainda ser utilizado na prática, HMAC é o mais aceito e utilizado. Portanto, neste trabalho, utilizaremos apenas HMAC como exemplo.

seguras entre si enquanto Eve não descobrir a chave secreta K.

O que acontece se Eve descobrir a chave secreta K? No momento em que Eve descobre a chave secreta K, as propriedades básicas de confidencialidade, integridade e autenticidade deixam de ter qualquer efeito para proteger tanto comunicações passadas quanto comunicações futuras entre Alice e Bob. Supondo que Eve esteve atuando de forma passiva na rede, ou seja, registrando todas as mensagens trocadas entre Alice e Bob, no momento em que Eve descobrir a chave secreta K, ela poderá decifrar e visualizar todas as mensagens passadas e futuras entre Alice e Bob. As primeiras perguntas que surgem são: *(a) Como podemos evitar que Eve consiga decifrar as mensagens das comunicações passadas entre Alice e Bob? (b) Como podemos evitar que Eve consiga decifrar as mensagens das comunicações futuras entre Alice e Bob?* Há duas propriedades avançadas de segurança que buscam respostas a estas duas perguntas. A primeira é PFS (*Perfect Forward Secrecy*), cujo objetivo é proteger a confidencialidade de comunicações passadas entre Alice e Bob. Já a segunda é PCS (*Post-Compromise Security*), que busca garantir a confidencialidade, integridade a autenticidade das mensagens de comunicações futuras entre Alice e Bob.

O principal objetivo deste minicurso é introduzir os conceitos e apresentar exemplos ilustrativos (simples e didáticos) e concretos de aplicação de PFS e PCS, que são propriedades de segurança avançadas e de grande importância na construção de sistemas mais resilientes à quantidade, força e inteligência crescente dos ataques modernos. Apesar de PFS ser um conceito relativamente antigo, na prática, são poucos os sistemas que realmente garantem esta importante propriedade de segurança. Por exemplo, no caso do TLS (*Transport Layer Security*) [Dierks and Rescorla 2008], apenas a versão mais recente (1.3 [Rescorla 2018]) garante, por padrão, PFS nas comunicações. Já PCS é algo bastante recente. Os primeiros trabalhos sobre o assunto começaram a surgir entre 2017 e 2018. Além disso, a complexidade e o número de mecanismos e recursos envolvidos em PCS é muito maior, como pode ser visto em exemplos práticos como DECIM [Yu et al. 2018], ART [Cohn-Gordon et al. 2018], RKE [Alwen et al. 2019, Poettering and Rösler 2018] e ANCHOR [Kreutz et al. 2017, Kreutz et al. 2019]. PCS envolve detecção, o que por si só é complicado, PCR (*Post-Compromise Recovery*) e mecanismos de geração e evolução de chaves futuras.

Este minicurso está organizado conforme segue. A Seção 2.1 aborda propriedades básicas de protocolos de segurança. Na sequência, as propriedades avançadas *perfect forward secrecy* e *post-compromise security* são apresentadas e discutidas nas Seções 2.2 e 2.3, respectivamente. A Seção 2.4 discorre sobre a importância de verificar a segurança de protocolos utilizando métodos formais e apresenta a ferramenta Scyther de verificação automática de protocolos. Finalmente, na Seção 2.5 são realizadas considerações finais acerca do material apresentado neste minicurso.

2.1. Projeto de protocolos de segurança

Os protocolos que protegem as comunicações devem fornecer as propriedades de confidencialidade, integridade e autenticidade dos dados em um cenário ideal. Tipicamente, estes protocolos incluem campos como um número de sequência, um *nonce* (também conhecido como valor único comumentemente utilizado para evitar ataques de *replay*

[Yang and Shieh 1999]), um *payload* (campo de dados) cifrado e um código de autenticação. Em boa parte dos sistemas, é comum as mensagens possuirem um tamanho padrão para evitar ataques que objetivam deduzir informação das comunicações a partir de tamanhos distintos das mensagens. Na prática, diferentes sistemas, como o Anonymizer, tem sido alvos de ataques (bem sucedidos) baseados no tamanho variado dos pacotes [Ling et al. 2013].

Pressupostos. Neste minicurso, para ilustrar os conceitos das propriedades básicas e avançadas de segurança, vamos utilizar um exemplo simples, para fins didáticos, de grupos de mensagens. Nós assumimos um servidor (Bob), um cliente (Alice) e um atacante (Eve). Bob gerencia N grupos de mensagens. Alice publica em grupos ou recebe mensagens dos grupos. Com o objetivo de simplificar as explicações e contribuir para a compreensão dos conceitos de PFS e PCS, consideraremos que Bob é confiável e seguro. Em outras palavras, assumimos que Eve consegue comprometer apenas os dispositivos e dados da Alice.

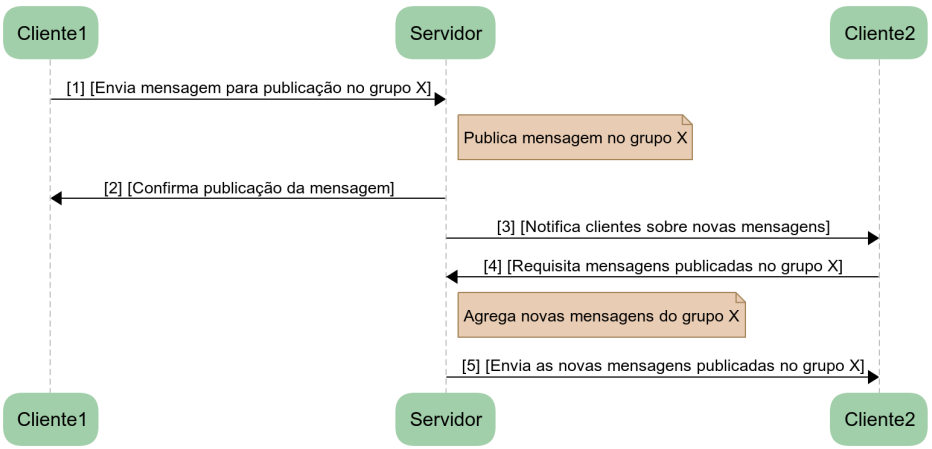


Figura 2.1: Serviço de grupos de mensagens.

A Figura 2.1 apresenta um fluxograma simples de troca de mensagens entre Alice (`Cliente1`) e Bob (`Servidor`) e Charles (`Cliente2`) e Bob. Como pode ser observado, Alice envia uma mensagem para publicação. O servidor recebe a mensagem e a publica no respectivo grupo. Depois de confirmar a publicação para o cliente, o servidor notifica o segundo cliente, interessado nas mensagens do grupo, sobre a existência de novas mensagens. O segundo cliente, por sua vez, requisita as mensagens publicadas no grupo. O servidor então envia todas as novas mensagens ao cliente.

É importante observarmos algumas coisas neste exemplo. Primeiro, as comunicações entre os clientes e servidor precisam garantir CIA. Isto pode ser alcançado através da utilização de uma chave secreta, compartilhada entre cada cliente e o servidor. Em outras palavras, o servidor terá uma lista de chaves secretas, uma para cada cliente. Segundo, é importante que as mensagens tenham o mesmo tamanho. Na prática, isto implica em segmentar os dados, isto é, limitar o tamanho máximo do *payload* da mensagem e, se

necessário, quebrar a mensagem original em várias partes. Por exemplo, considerando que o *payload* é de 512 bytes e a mensagem original do cliente tem 2048 bytes, a mensagem do cliente deverá ser enviada em quatro partes (quatro mensagens do protocolo de comunicação entre o cliente e o servidor). Por outro lado, se a mensagem original do cliente for menor que 512 bytes, o protocolo deverá realizar uma operação de *padding*, isto é, preencher a mensagem com "dados quaisquer" (definidos no protocolo) até completar 512 bytes. Dessa forma, todas as mensagens do protocolo, trocadas entre os clientes e o servidor, terão sempre um *payload* de 512 bytes.

A seguir, no Algoritmo 1, são apresentados os detalhes do protocolo utilizado entre os clientes e o servidor para envio, notificação e recuperação de mensagens. Como pode ser observado, existem apenas 5 comandos simples, `PUT`, `PUT_ACK`, `GET`, `GET_ACK` e `NOTIFY`. Como pode ser observado na linha 1, o cliente Alice envia uma nova mensagem para o servidor Bob utilizando o comando `PUT`, um *nonce*, um *payload* cifrado contendo a identificação do grupo e os dados da mensagem e uma assinatura HMAC. Bob (linha 2) envia uma mensagem de confirmação (`PUT_ACK`) para Alice. Alice e Bob compartilham uma chave secreta `Kab` para garantir as propriedades de confidencialidade, integridade e autenticidade das mensagens.

Em seguida, Bob envia uma mensagem de notificação (`NOTIFY`) para Charles, avisando que há mensagens novas no(s) grupo(s) de interesse (linha 3). Charles escolhe o grupo e envia uma solicitação (`GET`) para o servidor (linha 4). O servidor responde (`GET_ACK`) com a identificação do grupo e o conteúdo das mensagens que couberem no *payload*. Novamente, os dados entre Charles e Bob são protegidos (CIA) através de uma chave secreta compartilhada `Kcb`.

Algoritmo 1: Comunicação entre os clientes (Alice e Charles) e o servidor (Bob)

1.	Alice $\rightarrow$ Bob	[PUT, *nonce*, E_{Kab}(grupo, mensagem)], HMAC$_{Kab}$
2.	Bob $\rightarrow$ Alice	[PUT_ACK, *nonce*, E_{Kab}(*nonce*)], HMAC$_{Kab}$
3.	Bob $\rightarrow$ Charles	[NOTIFY, *nonce*, E_{Kcb}(lista_de_grupos)], HMAC$_{Kcb}$
4.	Charles $\rightarrow$ Bob	[GET, *nonce*, E_{Kcb}(grupo)], HMAC$_{Kcb}$
5.	Bob $\rightarrow$ Charles	[GET_ACK, *nonce*, E_{Kcb}(grupo, mensagem)], HMAC$_{Kcb}$

O *nonce* pode ser gerado de forma simples e eficiente (e.g. $nonce = idvv\_next()$), utilizando o gerador de iDVVs (*integrated Device Verification Values*) proposto por pesquisa recente [Kreutz et al. 2018, Kreutz et al. 2017]. O gerador de iDVVs pode ser utilizado de forma a garantir a geração de *nonce* de alta qualidade com um baixo custo computacional.

O nosso pressuposto é que a função de geração do *nonce* gera um valor único para cada mensagem, proporcionando robustez ao protocolo. O iDVV é um gerador que une a indistinguibilidade de geradores pseudo-aleatórios e as propriedades de números determinísticos, gerando a mesma sequência em ambas as extremidades do canal de comunicação. Isto permite a geração e a verificação descentralizada e segura dos valores utilizados (e.g., *nonce*) entre o cliente e o servidor [Kreutz et al. 2018].

Resumidamente, enquanto as chaves secretas compartilhadas entre os clientes e o servidor (`Kab` e `Kcb`) não forem descobertas pelo agente malicioso Eve, as mensagens trocadas estão protegidas e possuem as propriedades de confidencialidade, integridade e autenticidade asseguradas. Entretanto, é possível que Eve seja um agente malicioso ativo que registra todo o tráfego de mensagens da rede entre o cliente Alice e o servidor Bob, como ilustrado no diagrama da Figura 2.2. Neste caso, no momento em que Eve descobrir a chave secreta `Kab`, o agente malicioso terá acesso aos dados originais (decifrados) de todas as comunicações passadas, presentes e futuras da Alice com Bob. Ao observar cuidadosamente a figura, pode-se perceber que o comprometimento da chave secreta `Kab` compromete também as comunicações entre o cliente Alice (1) e o cliente Charles (2), realizadas através do grupo X no servidor.

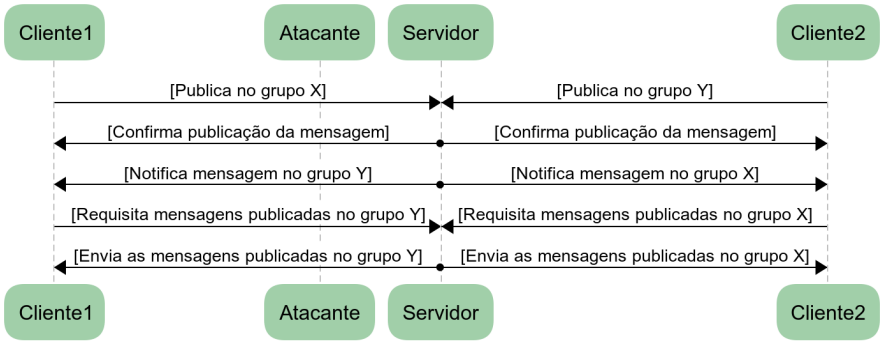


Figura 2.2: Grupos de mensagens com atacante ativo

O atacante pode descobrir a chave secreta `Kab` de diferentes formas, como comprometimento do dispositivo do cliente 1 (exemplo: através de *trojans* ou *keyloggers*), *side-channel attacks* [Meyer et al. 2014, Zhang et al. 2014], ataques de engenharia social [Krombholz et al. 2015, Thornburgh 2004], *remote timing attacks* [Brumley and Tuveri 2011], entre outros. A pergunta que surge é: *Como proteger as comunicações passadas, presentes e futuras em caso de comprometimento da chave secreta utilizada entre um cliente e o servidor?* Há propriedades avançadas de segurança, como PFS e PCS, que tem por objetivo garantir a segurança das comunicações passadas e futuras. Com relação às comunicações presentes, realizadas durante o intervalo do ataque, não há o que fazer, isto é, o atacante terá acesso às mensagens em trânsito até que o ataque seja detectado pelo cliente ou pelo servidor. Uma vez detectado o ataque, protocolos de PCR (*Post-Compromise Recovery*) e PCS (*Post-Compromise Security*) podem ser ativados para garantir a CIA das comunicações futuras.

2.2. Perfect Forward Secrecy

Perfect forward secrecy é uma propriedade que garante a segurança de comunicações passadas, isto é, comunicações realizadas antes do início do ataque. Uma forma bastante simples de ilustrar o princípio de PFS é apresentada na Figura 2.3. Como pode ser observado, há três sessões de comunicação entre o cliente o servidor. Em cada sessão, é utilizada uma chave secreta distinta (`kc1A`, `kc1B` e `kc1C`). Desta forma, se o atacante

comprometer a chave de sessão `kc1C`, ele/ela compromete apenas os dados da terceira sessão, mas não das duas sessões anteriores. Este é o princípio essencial por trás do PFS. Claro que, na prática, é, geralmente, um pouco mais complicado que isto garantir *perfect forward secrecy* em um protocolo de comunicação.

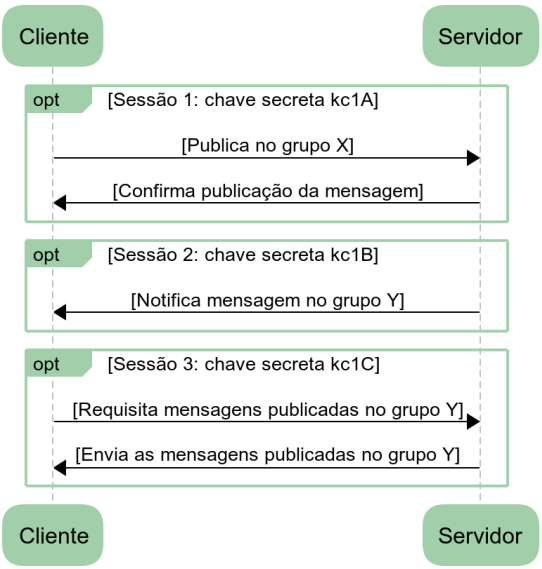


Figura 2.3: Sessões de comunicação entre o cliente e o servidor

O PFS é utilizado para prevenir que o comprometimento de uma chave secreta de longo termo seja usado para afetar a confiabilidade de comunicações anteriores ao ataque. Se uma única comunicação for comprometida, isso não compromete a confidencialidade de todas as comunicações anteriores a ela. O conceito de PFS pode ser aplicado em qualquer protocolo. O mecanismo mais aceito e utilizado é a rotação e atualização constante de chaves. A rotação de chaves pode ser realizada com base em diferentes parâmetros:

(p_1) Uma chave única para toda a comunicação entre o cliente e o servidor (e.g. Figura 2.1);

(p_2) Uma chave por sessão de comunicação (e.g. Figura 2.3), onde a sessão é definida por uma unidade de tempo;

(p_3) Uma chave para cada mensagem trocada entre o cliente e o servidor.

Neste minicurso nós iremos utilizar chaves por sessão de comunicação. Uma sessão pode ser definida por um intervalo de tempo x, como ilustrado na representação matemática 1 [Arsenault 2017].

$$[a,b] = \{x \in \mathbb{R} \| a \leq x \leq b\} \tag{1}$$

Voltando ao exemplo da Figura 2.1, ao utilizar a idéia de chaves por sessão, temos um novo fluxograma conforme ilustrado na Figura 2.4. Como pode ser observado, há uma

sessão de comunicação N entre o cliente e o servidor. Supondo que cada sessão possui uma chave única, caso o atacante descubra a(s) chave(s) da sessão N, ele conseguirá comprometer as comunicações desta sessão. Entretanto, apenas com as chaves da sessão N, não será possível comprometer os dados das sessões anteriores, ou seja, das sessões 1 até N-1.

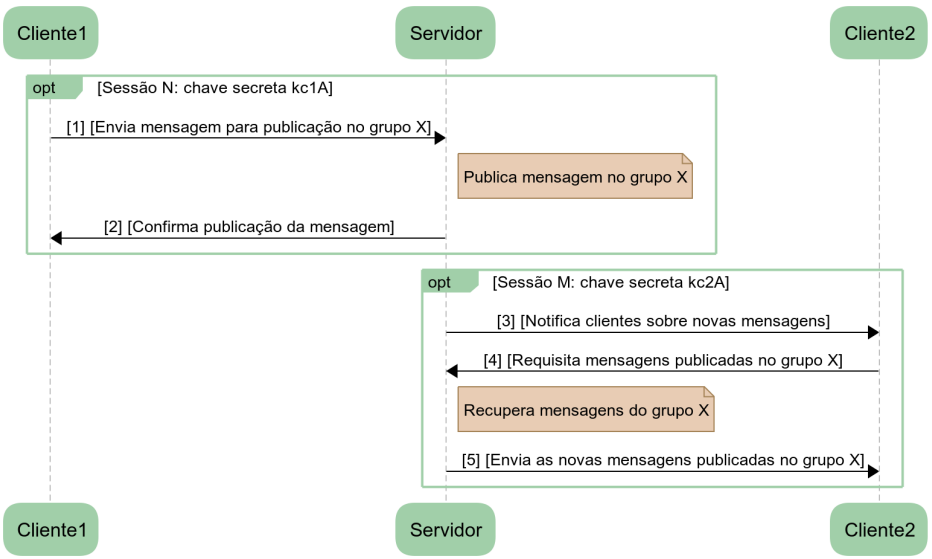


Figura 2.4: Sessões de comunicação entre os clientes e o servidor.

Uma forma simples de evoluir uma chave, sem possibilidade de reversão (isto é, descobrir a chave anterior a partir da atual), é através de funções de hash criptográficas seguras (e.g. SHA-256 e SHA-512), como proposto e utilizado em protocolos atuais que garantem PFS [Kreutz et al. 2019]. No Algoritmo 2, é possível observar como é gerada a chave de longa duração (chave inicial) e as respectivas chaves de sessão.

Primeiramente, os clientes (Alice e Charles) e o servidor (Bob) geram uma chave de longa duração através de um algoritmo como o Diffie-Hellman (DH) [Rescorla 1999]. A partir da chave de longa duração (K_{CLD}) é derivada a primeira chave secreta de sessão (K_{css}). A chave secreta de sessão é atualizada a cada intervalo de tempo x, como ilustrado nas linhas 3, 5 e 8 do algoritmo. Tanto a primeira chave, quanto as atualizações, são geradas utilizando uma função de hash criptográfica H, que recebe como parâmetro a chave de longa duração ou a chave de sessão, respectivamente.

Supondo que o atacante comprometa a chave K_{css} da linha 8, as chaves de sessão das linhas 3 e 5 estão seguras, pois estas já foram sobrescritas e, por definição de construção, não há como reverter o resultado (*digest*) de uma função de hash criptográfica H (e.g., SHA-256). Como resultado, o atacante não consegue decifrar as mensagens anteriores ao comprometimento da chave de sessão. No entanto, vale ressaltar que PFS não protege o sistema contra ataques de criptoanálise. Um ataque deste tipo consiste em encontrar uma maneira de decifrar uma mensagem cifrada sem a chave secreta. Um técnica básica de

Algoritmo 2: Geração e atualização de chaves secretas de sessão.

	Alice $\overset{\text{DH}}{\longleftrightarrow}$ Bob	Geração da chave de longa duração K_{CLD}
	Charles $\overset{\text{DH}}{\longleftrightarrow}$ Bob	Geração da chave de longa duração K_{CLD}
	Alice, Bob, Charles	Deriva 1a Chave Secreta de Sessão: $K_{css} \leftarrow \text{H}(K_{CLD})$
1.	Alice $\rightarrow$ Bob	[`PUT`, *nonce*, E(grupo, mensagem)], HMAC
2.	Bob $\rightarrow$ Alice	[`PUT_ACK`, *nonce*, E(*nonce_prev*)], HMAC
3.	Alice, Bob	Atualização da Chave Secreta da Sessão: $K_{css} \leftarrow \text{H}(K_{css})$
4.	Bob $\rightarrow$ Charles	[`NOTIFY`, *nonce*, E(lista_de_grupos)], HMAC
5.	Bob, Charles	Atualização da Chave Secreta da Sessão: $K_{css} \leftarrow \text{H}(K_{css})$
6.	Charles $\rightarrow$ Bob	[`GET`, *nonce*, E(grupo)], HMAC
7.	Bob $\rightarrow$ Charles	[`GET_ACK`, *nonce*, E(grupo, mensagem)], HMAC
8.	Bob, Charles	Atualização da Chave Secreta da Sessão: $K_{css} \leftarrow \text{H}(K_{css})$

criptoanálise consiste em identificar as letras que aparecem com mais frequência em uma linguagem e associá-las as letras que aparecem com maior frequência no texto cifrado.

Há um aspecto importante a ser observado no Algoritmo 2. No início do algoritmo, é gerada uma chave de longa duração K_{CLD}. *O que acontece se o atacante comprometer esta chave?* Este é um exemplo simples, porém real, de pressupostos fracos existentes em sistemas. Se um atacante ativo comprometer a chave K_{CLD}, ele é capaz de comprometer a confidencialidade das comunicações cujas chaves secretas foram derivadas da chave de longa duração atual. Uma forma simples de resolver o problema é simplesmente apagar a chave K_{CLD} logo após o primeiro uso.

PFS: Exemplos de Uso Prático

Quando o protocolo de comunicação oferece a propriedade de segurança PFS, um atacante, mesmo tendo acesso a chave secreta da sessão de tempo t_1, não conseguirá decifrar as mensagens trocadas nos instantes de tempo t_2 e t_3, onde $t_1 > t_2 > t_3$. O TLS 1.3 [Rescorla 2018] é a primeira versão do TLS a efetivamente oferecer PFS. Outros protocolos e modelos de segurança, como o IKEv1 [Hoffman 2005], email seguro [Sun et al. 2005, Kim et al. 2006], 5G AKA [Arkko et al. 2015] e eCK [Cremers and Feltz 2012], também oferecem PFS. Além disso, mais recentemente, PFS tem sido utilizado em alguns sistemas como mecanismo para evitar ataques de retransmissão [Zenger et al. 2016].

No caso do TLS, em cada conexão, a chave de sessão efêmera, que não depende do uso de certificado do server, é rotada. Para oferecer PFS, essa chave não pode ser guardada e nem reutilizada. Dessa forma, uma chave privada de sessão efêmera, que tenha sido comprometida, não tem utilidade alguma para decifrar mensagens de sessões efêmeras passadas [Bernat 2011, Rutishauser 2017].

2.3. Post-Compromise Security

Enquanto PFS protege as comunicações passadas, *Post-Compromise Security* (PCS) tem por objetivo proteger as comunicações futuras, isto é, aquelas ocorridas depois do ataque. Para que isto seja possível, primeiro, é necessário detectar que houve um comprometimento da chave secreta compartilhada entre o cliente Alice e o servidor Bob, como ilustrado no diagrama da Figura 2.5. Um dos primeiros trabalhos a propor uma forma de detectar o comprometimento da chave secreta foi o DECIM [Yu et al. 2018]. Vale ressaltar que o processo não é simples e nem fácil de ser implementado. No caso do DECIM, há, inclusive, a necessidade de intervenção manual do usuário, que recebe um alerta, gerado a partir da análise de logs públicos, de algo anormal no sistema.

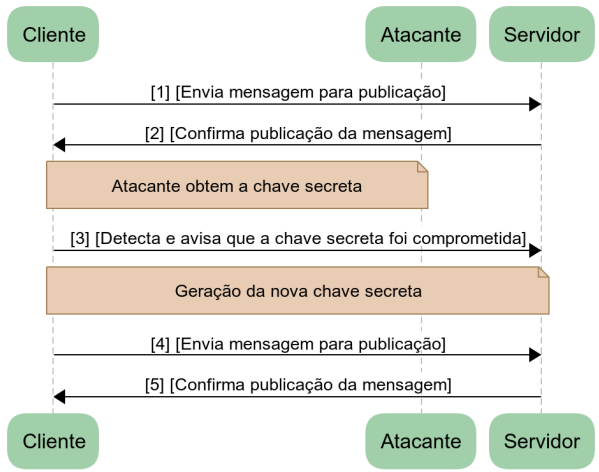


Figura 2.5: Chave comprometida e posterior recuperação do sistema

Além da detecção, pode ser necessária a recuperação das chaves para voltar a estabelecer as comunicações, de forma segura, com as diferentes entidades, como proposto em ANCHOR através de um protocolo de PCR (*Post-Compromise Recovery*) [Kreutz et al. 2019]. Neste minicurso, iremos considerar um oráculo capaz de "magicamente" detectar o comprometimento da chave secreta utilizada nas comunicações entre Alice e Bob (cliente e servidor). Além disso, vamos utilizar um protocolo de PCR simplificado, baseado na utilização do algoritmo de geração de chaves de Diffie-Hellman (DH). Isto é o suficiente para ilustrar a ideia e os conceitos de PCS.

O diagrama da Figura 2.6 ilustra uma sequência de comunicações entre o Alice (cliente) e Bob (servidor) onde há atualização de chaves (para garantir PFS), um ataque que compromete o cliente e a chave secreta, a execução de um protocolo PCS e a volta à normalidade do sistema, isto é, continua a garantir a CIA das mensagens. Como pode ser observado, nas etapas 4 e 5, o atacante explora uma vulnerabilidade e compromete o cliente e a chave secreta. Com a detecção do comprometimento, é ativado o protocolo de PCS, que, neste exemplo simples, engloba a remoção da vulnerabilidade (e.g., atualização de software) e a geração de uma nova chave secreta compartilhada utilizando o DH. Na sequência, o sistema volta a operar normalmente e o conteúdo das mensagens futuras

(após a execução do protocolo de PCS) não pode mais ser comprometido pelo atacante, visto que a chave secreta comprometida já não vale mais no sistema.

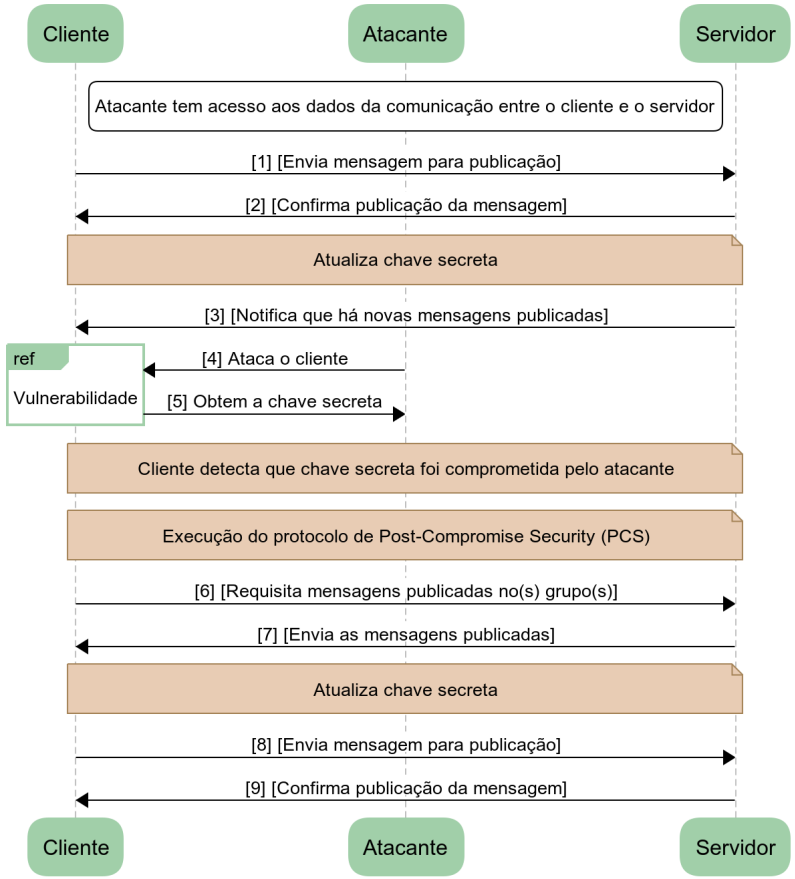


Figura 2.6: Alice e sua chave secreta são comprometidos pelo atacante

No Algoritmo 3 é detalhado o exemplo com PCS. O cliente Alice e o servidor Bob iniciam gerando uma chave secreta de sessão K_{css}, que é utilizada na proteção da CIA das mensagens trocadas na sessão. Na linha 3, a chave de sessão é atualizada utilizando uma função de hash criptográfica H. Nas linhas 5 e 6, o atacante compromete Alice, conseguindo acesso à chave secreta de sessão e Alice detecta o comprometimento, respectivamente.

Alice e Bob executam então o protocolo de recuperação e geração da nova chave secreta de sessão (linhas 7 e 8). Como comentado anteriormente, a chave é atualizada utilizando DH, o que não garante resiliência contra ataques de computadores quânticos, isto é, não é PQS (*Post-Quantum Secure*), mas é o suficiente para os dias atuais. Na sequência, continua o protocolo normalmente com a nova chave K_{css}. Por fim, a chave é novamente atualizada (linha 11).

Algoritmo 3: Comunicação com *Post-Compromise Security*

	Alice, Bob	Geração da chave secreta de sessão K_{css}
1.	Alice $\rightarrow$ Bob	[`PUT`, *nonce*, E(grupo, mensagem)], HMAC
2.	Bob $\rightarrow$ Alice	[`PUT_ACK`, *nonce*, E(*nonce*)], HMAC
3.	Bob, Alice	K_{css} = H(K_{css});
4.	Bob $\rightarrow$ Alice	[NOTIFY, *nonce*, E(lista_de_grupos)], HMAC
5.	Alice	É comprometido pelo atacante.
6.		Detecta que chave secreta de sessão foi comprometida.
7.	Alice $\overset{\text{PCS}}{\longleftrightarrow}$ Bob	Execução de protocolo de recuperação PCS.
8.		Geração da nova chave secreta de sessão K_{css}
9.	Alice $\rightarrow$ Bob	[`PUT`, *nonce*, E(grupo, mensagem)], HMAC
10.	Bob $\rightarrow$ Alice	[`PUT_ACK`, *nonce*, E(*nonce*)], HMAC
11.	Bob, Alice	K_{css} = H(K_{css});

O Algoritmo 4 ilustra o funcionamento do método de Diffie-Helmann. A chave resultante é a potência dos valores (a e b) trocados entre Alice e Bob, conforme detalhado no algoritmo. Alice gera e envia para Bob o valor a (linhas 3 e 6). Da mesma forma, Bob gera e envia para Alice o valor b (linhas 4 e 7). Ambos calculam a chave secreta compartilhada K através da equação $g^{AB}(mod p)$ (linhas 8 e 9). Para aumentar a segurança, Alice deriva uma nova chave K_s utilizando uma função de derivação HKDF (linha 10) [Krawczyk and Eronen 2010]. Alice envia a nova chave para Bob (linha 11), ambos atualizam a chave K (linha 12) e, por fim, Bob envia uma mensagem para Alice para confirmar que atualizou e está utilizando a nova chave secreta (linha 13). Para isso, Bob envia para Alice o $nonce_s$, recebido na linha 11, cifrado com a nova chave.

No caso do DH, para aumentar a segurança do algoritmo e da chave compartilhada, é importante garantir que o número primo possua aproximadamente 300 dígitos e os números secretos possuam pelo menos 100 dígitos cada [Rescorla 1999, Kivinen and Kojo 2003]. Desta forma, o atacante será obrigado a enfrentar o problema do logaritmo discreto, inviabilizando um ataque de força bruta contra o algoritmo de Diffie-Helmann. O diagrama da Figura 2.7 ilustra o processo de recuperação de chave secreta de sessão segundo o Algoritmo 4.

Como assumimos que o servidor Bob é confiável e capaz de garantir a segurança das chaves mestras dos clientes Alice e Charles, podemos realizar a recuperação (em caso de comprometimento) utilizando apenas criptografia simétrica. Com isso, o sistema torna-se também PQS, isto é, não dependente do DH. Supondo que Alice e Bob possuem uma chave mestra secreta (K_{cms}), compartilhada. Como o cliente pode ser comprometido, a chave K_{cms} deve ser armazenada off-line pelo cliente (e.g., num pendrive ou anotado num papel e guardado na carteira). Esta é uma prática cada vez mais comum em serviços de

Algoritmo 4: Recuperação de chaves utilizando Diffie-Helmann

1.	Alice, Bob	Definição dos parâmetros p e g
2.	Alice	$A = random()$
3.		$a = g^A (mod p)$
4.	Bob	$B = random()$
5.		$b = g^B (mod p)$
6.	Alice $\rightarrow$ Bob	[DH_A, a, $nonce$]
7.	Bob $\rightarrow$ Alice	[DH_B, b, $nonce$]
8.	Alice	$K = g^{BA} (mod p) = b^A (mod p)$
9.	Bob	$K = g^{AB} (mod p) = a^B (mod p)$
10.	Alice	$K_s = \text{HKDF}(K, l, s, i)$
11.	Alice $\rightarrow$ Bob	[DH_KEY, $nonce$, E(K_s, $nonce_s$)], HMAC
12.	Alice, Bob	$K = \text{H}(K_s)$;
13.	Bob $\rightarrow$ Alice	[DH_KEY, $nonce$, E($nonce_s$)], HMAC

segurança internacionais (e.g., recuperação online de senhas em bancos como o BCEE[8]) e empresas de armazenamento seguro de dados similares a BlueFiles[9] e CryptoBox[10].

Com criptografia simétrica, após o comprometimento da Alice (linhas 5 e 6 do Algoritmo 3), a recuperação envolve a utilização da chave mestra secreta, que é armazenada off-line. Como proposto em trabalhos recentes [Kreutz et al. 2017, Kreutz et al. 2019], a chave mestra do cliente pode ser utilizada apenas off-line para computar a nova chave secreta de sessão (linhas 7 e 8). O diagrama da Figura 2.8 e o Algoritmo 5 detalham o processo de recuperação e geração de uma nova chave secreta de sessão a partir de uma chave mestra secreta off-line.

Primeiro, Alice computa off-line a chave de recuperação K_{rec} a partir da chave mestra secreta K_{cms} (linhas 1 a 3 do Algoritmo 5). Em seguida, envia o valor do *random* utilizado para derivar a chave K_{rec} para Bob. Bob deriva a chave K_{rec} (linha 5) e verifica a integridade e autenticidade da mensagem através do HMAC gerado com a nova chave de recuperação. O atacante não tem como forjar a mensagem sem ser detectado pelo fato de não conhecer a chave K_{cms} utilizada na derivação da chave K_{rec}. Bob então envia uma mensagem de confirmação para Alice (linha 6). Finalmente, ambos atualizam a chave secreta de sessão K_{css} (linha 7) e confirmam o conhecimento da nova chave (linha 8).

[8] https://www.bcee.lu
[9] https://mybluefiles.com
[10] https://www.ercom.com/cryptobox/

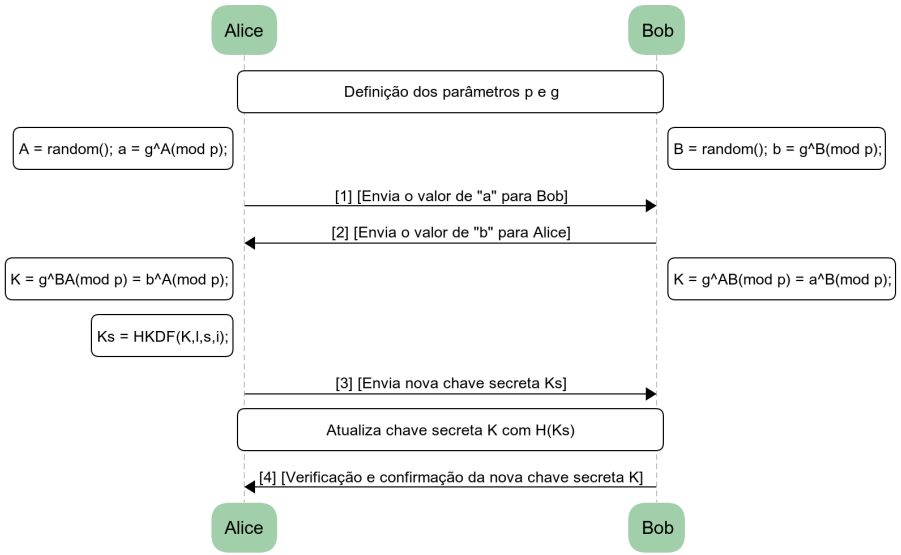


Figura 2.7: Recuperação de chave secreta de sessão com DH.

Algoritmo 5: Recuperação de chaves utilizando criptografia simétrica

1.	Alice	Computa chave de recuperação K_{rec} off-line.
2.		$random = idvv\_next()$;
3.		$K_{rec} = \mathrm{H}(K_{cms}\|\|random)$;
4.	Alice $\rightarrow$ Bob	[REC, *nonce*, *random*], HMAC
5.	Bob	$K_{rec} = \mathrm{H}(K_{cms}\|\|random)$;
6.	Bob $\rightarrow$ Alice	[REC_ACK, *nonce*, E(nonce)], HMAC
7.	Alice, Bob	$K_{css} = \mathrm{H}(K_{rec})$;
8.	Bob $\overset{\text{ACK}}{\longleftrightarrow}$ Alice	[KEY, *nonce*, E(nonce)], HMAC

PCS: Exemplos de Uso Prático

O número de sistemas e protocolos com suporte a PCS ainda é significativamente limitado. A principal razão é o fato de PCS ser algo recente e em pleno estado de investigação e desenvolvimento. Um dos primeiros papers sobre o assunto foi publicado em 2016 [Cohn-Gordon et al. 2016]. De lá para cá, alguns protocolos e sistemas com propriedades de segurança avançada, começaram a incluir PCR (*Post-Compromise Recovery*, parte necessária à PCS) e PCS no seu projeto [Yu et al. 2018, Lehmann and Tackmann 2018, Kreutz et al. 2017, Kreutz et al. 2019, Cohn-Gordon et al. 2018, Alwen et al. 2019, Poettering and Rösler 2018].

DECIM (*Detecting Endpoint Compromise in Messaging*) [Yu et al. 2018] foi um dos primeiros sistemas a efetivamente propor meios de detectar o comprometimento de

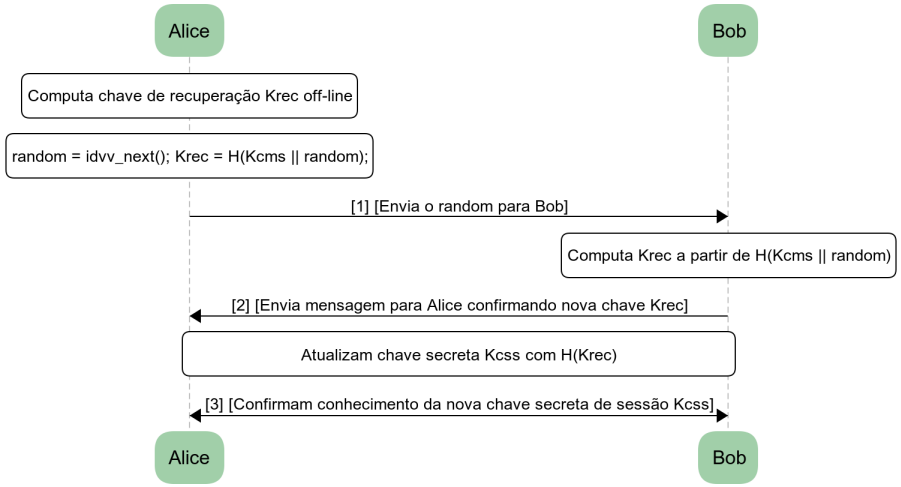


Figura 2.8: Recuperação de chave secreta de sessão com criptografia simétrica.

endpoints em sistemas de comunicação instantânea (e.g., WhatsApp). DECIM é capaz de gerenciar e atualizar chaves criptográficas de uma maneira automática e transparente. Para isso, é necessário registrar os usos das chaves em serviços de log, que estão publicamente disponíveis e podem ser consultados pelos usuários para identificar potenciais usos indevidos de chaves. Segunda estatísticas reportadas, o sistema de mensagens DECIM é eficiente mesmo para milhões de usuários. Durante a operação do DECIM, o dispositivo do destinatário certifica automaticamente novos pares de chaves, armazenando os certificados em um log público que garante a integridade dos dados lá armazenados.

Devido também aos recorrentes e recentes incidentes de segurança, mais especificamente vazamentos de dados sensíveis ligados a aplicativos de mensagem instantânea, PFS está em alta e sendo aplicado na prática em sistemas de comunicação instantânea e comunicação de grupos, como o Signal[11], o WhatsApp[11] e o Telegram[11]. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lo-

[11] https://www.signal.org/, https://www.whatsapp.com/, https://telegram.org/.

rem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Recentemente, pesquisadores vem investigado e propondo novos protocolos (e.g., ART [Cohn-Gordon et al. 2018]), ou novas versões de protocolos (e.g., RKE [Alwen et al. 2019, Poettering and Rösler 2018]), cada vez mais robustos com relação a PCS, uma propriedade de segurança importante e necessária nos sistemas atuais.

2.4. Verificação Automática de Protocolos

A verificação automática de propriedades de segurança é, hoje em dia, imprescindível para verificar a corretude de protocolos de segurança [Chudnov et al. 2018, Li et al. 2018, Kreutz et al. 2019, Jenuario et al. 2019]. Apesar de pouco conhecidas [Jenuario et al. 2019], ferramentas como a Scyther [Cremers 2006] contribuem nesse processo.

Em [Jenuario et al. 2019], os autores apresentam de forma didática e simples uma introdução à semântica e funcionamento da Scyther. O objetivo desta seção é demonstrar a representação semântica e o processo de análise e verificação de possíveis falhas de segurança do protocolo PFS proposto no Algoritmo 2 da Seção 2.2.

O Algoritmo 6 apresenta o Algoritmo 2 (PFS) na semântica da ferramenta Scyther. As chaves de sessão K, Kir e Kr, que representam as chaves K_{CLD}, K_{css} e K_{css} do algoritmo original, respectivamente, são declaradas globalmente (linha 1). As chaves Kir e Kr são derivadas da chave K. Como a ferramenta Scyther não possui funções próprias para cifrar e decifrar dados utilizando criptografia simétrica, a função global `H` (linha 2) é utilizada para representar a cifragem de termos específicos no algoritmo.

A definição `HMAC`, precedida de *const* e definida como sendo do tipo **Function** (linha 3), determina que o é uma função global que aceita diferentes termos como parâmetro de entrada. No caso do algoritmo, `HMAC` é utilizado como meio de gerar e verificar a assinatura dos dados enviados entre `Alice` e `Bob`.

A chamada à função **protocol** (linha 6) marca o início da especificação do protocolo denominado `PFS`, com cinco agentes, sendo que os quatro primeiros (`KGC`, `KDF`, `Alice` e `Bob`) possuem papéis (**role**) explícitos, enquanto que *Eve* (linha 4) não possui. Por definição, a Scyther implementa implicitamente o papel do agente não confiável, no caso, *Eve*.

Como pode ser observado na semântica do algoritmo, cada evento de envio (e.g., **send_1**) possui um evento de recebimento (e.g., **recv_1**) correspondente (e.g., linhas 8 e 11). A sintaxe do evento **send_1** indica que a transmissão é de `KGC` (*Key Generation Center*) para `KDF` (*Key Deverivation Function*), simulando a geração da chave de longa duração K, cifrada com a função de hash criptográfica `H`.

No agente `KDF` (linha 11), há um evento com a sintaxe idêntica ao `KGC`, cuja única diferença é o tipo, i.e., **recv** ao invés de **send**, inicializando a chave K de longa duração. Ainda no agente `KDF` (linhas 12 e 13), é gerada a primeira chave de sessão Kir, que é

Algoritmo 6: Algoritmo 2 (PFS) na semântica da Scyther

```
1  secret K, Kir, Kr: SessionKey;
2  hashfunction H;
3  const HMAC: Function;
4  const Eve: Agent;
5  unstrusted Eve;
6  protocol PFS(KGC,KDF,Alice,Bob,Eve){
7     role KGC{
8        send_1(KGC,KDF,H(K));
9     }
10    role KDF{
11       recv_1(KGC,KDF,H(K));
12       send_2(KDF,Alice,K(Kr));
13       send_3(KDF,Bob,K(Kr));
14       send_6(KDF,Alice,K(Kir));
15       send_7(KDF,Bob,K(Kir));
16    }
17    role Alice{
18       fresh nonce, grupo, mensagem: Nonce;
19       var nonceprev, listadegrupos: Nonce;
20       recv_2(KDF,Alice,K(Kr));
21       send_4(Alice,Bob,
      HMAC(nonce,{grupo,mensagem}Kr(Alice,Bob)));
22       recv_5(Bob,Alice, HMAC(nonce,{nonceprev}Kr(Bob,
      Alice)));
23       recv_6(KDF,Alice,K(Kir));
24       recv_8(Bob, Alice, HMAC(nonce,{listadegrupos}
25    Kir(Bob, Alice)));
26       claim(Alice,Secret,K);
27       claim(Alice,Secret,grupo);
28       claim(Alice,Secret,mensagem);
29       claim(Alice,Secret,nonceprev);
30       claim(Alice,Secret,listadegrupos);
31       claim(Alice,Secret,Kir);
32    }
33    role Bob{
34       fresh nonceprev, listadegrupos: Nonce;
35       var nonce, grupo, mensagem: Nonce;
36       recv_3(KDF,Bob,K(Kr));
37       recv_4(Alice,Bob,
      HMAC(nonce,{grupo,mensagem}Kr(Alice,Bob)));
38       send_5(Bob,Alice, HMAC(nonce,{nonceprev}
39    Kr(Bob, Alice)));
40       recv_7(KDF,Bob,K(Kir));
41       send_8(Bob, Alice, HMAC(nonce,
42    {listadegrupos}Kir(Bob, Alice)));
43       claim(Bob,Secret,K);
44       claim(Bob,Secret,grupo);
45       claim(Bob,Secret,mensagem);
46       claim(Bob,Secret,nonceprev);
47       claim(Bob,Secret,listadegrupos);
48       claim(Bob,Secret,Kir);
49    }
50 }
```

derivada da chave de longa duração K e inicializada através dos eventos de **send** e **recv** (linhas 20 e 36), garantindo que a chave de sessão seja a mesma para os agentes `Alice` e `Bob`.

Após a inicialização da chave de sessão na agente `Alice`, são criadas as variáveis *nonce*, *grupo* e *mensagem* (linha 18), do tipo **Nonce** e antecedida por **fresh**, o que significa que essas variáveis irão conter um valor pseudo-aleatório (e.g., linha 21). Já as variáveis *nonceprev* e *listadegrupos* (linha 19), são do mesmo tipo, mas **var** ao invés de **fresh**, o que significa que a variável será utilizada para armazenar valores recebidos (e.g., linhas 22 e 24).

Para determinar se um termo é secreto (**Secret**), é necessário que hajam eventos de afirmação (i.e., **claim**, como pode ser observado nas linhas 26 a 31 e 43 a 48) que definem os requisitos de segurança do protocolo. No caso, as afirmações criadas verificam se os *nonces* gerados por `Alice` (*nonce*, *grupo*, *mensagem*) e por `Bob` (*nonceprev*, *listadegrupos*), bem como a chave de longa duração K e as de sessão Kr e Kir, permanecem secretos durante as comunicações.

Ao analisar um protocolo na ferramenta Scyther, é gerado um relatório que aponta se existem falhas no protocolo. Quando há falhas, a ferramenta apresenta também um fluxograma que ilustra em detalhes como o ataque pode ser realizado ao protocolo [Jenuario et al. 2019]. Para o caso do código do Algoritmo 6, como pode ser observado na Figura 2.9, a comunicação entre `Alice` e `Bob` é segura segundo a análise automática da ferramenta, pois não há nenhum indicativo de falha no *Status* do relatório (i.e., tudo está como *Ok*, verificado e confirmado como sem ataques).

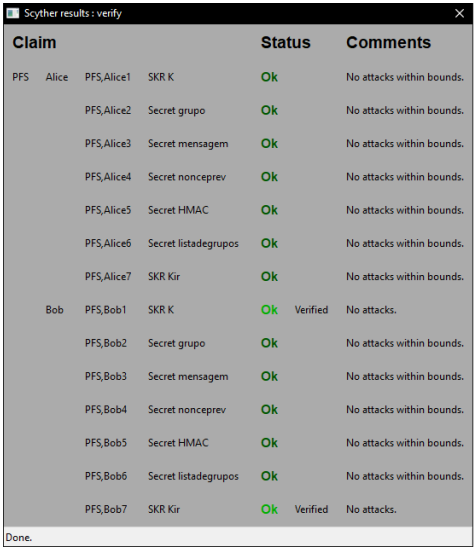


Figura 2.9: Verificação do Algoritmo 6 (PFS)

Para observar o comportamento de um protocolo, pode-se definir propositalmente um agente não-confiável *Eve* que é capaz de comprometer as chaves utilizadas na comunicação dos demais agentes, como `Alice` e `Bob`, por exemplo. Nesse caso, tomando como exemplo o Algoritmo 6, a ferramenta considera a existência de um agente malicioso que conhece as chaves secretas de comunicação de `Alice` e `Bob`.

Assumindo um agente malicioso *Eve*, as comunicações entre `Alice` e `Bob` podem ser comprometidas de três formas. A primeira é quando o atacante compromete a chave de longa duração ("*compromised Eve*(K);", na semântica da ferramenta), resultando no comprometimento de todas as chaves e comunicações entre `Alice` e `Bob`. As outras duas formas de comprometer parte das comunicações é através do comprometimento das chaves de sessão ("*compromised Eve*(Kr);", na semântica da ferramenta) ou ("*compromised Eve*(Kir);", na semântica da ferramenta), respectivamente. Nestes casos, serão comprometidas apenas as comunicações das respectivas sessões e não de todas as comunicações, como é o caso do comprometimento da chave K.

2.5. Considerações Finais

Este minicurso apresentou os conceitos básicos de segurança, aqui denominados de CIA (Confidencialidade, Integridade e Autenticidade), de uma forma simples, prática e didática. Além disso, progressivamente, foram discutidas duas propriedades avançadas de segurança, *Perfect Forward Secrecy* (PFS) e *Post-Compromise Security* (PCS), que são pouco conhecidas e estudadas por entusiastas de tecnologia, estudantes de computação e profissionais da área de tecnologia de um modo geral.

Resumidamente, o minicurso atingiu o seu principal objetivo, isto é, contribuir com a disseminação desses importantes e atuais temas da área de segurança da informação. Em um cenário cada vez mais agressivo em termos de ataques cibernéticos sofistica-

dos, é essencial prezarmos pelo conhecimento e formação de pessoas no que diz respeito à segurança da informação.

Referências

[Alwen et al. 2019] Alwen, J., Coretti, S., and Dodis, Y. (2019). The double ratchet: security notions, proofs, and modularization for the signal protocol. In *Int. Conf. on the Theory and Applications of Cryptographic Techniques*, pages 129–158.

[Arkko et al. 2015] Arkko, J., Norrman, K., Näslund, M., and Sahlin, B. (2015). A usim compatible 5g aka protocol with perfect forward secrecy. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 1205–1209. IEEE.

[Arsenault 2017] Arsenault, C. (2017). Perfect Forward Secrecy – Why You Should Be Using It. http://bit.do/pfs-101.

[Bernat 2011] Bernat, V. (2011). TLS & Perfect Forward Secrecy. http://bit.do/pfs-102.

[Brumley and Tuveri 2011] Brumley, B. B. and Tuveri, N. (2011). Remote timing attacks are still practical. In *ESORICS*, pages 355–371.

[Chudnov et al. 2018] Chudnov, A., Collins, N., Cook, B., Dodds, J., Huffman, B., MacCárthaigh, C., Magill, S., Mertens, E., Mullen, E., Tasiran, S., Tomb, A., and Westbrook, E. (2018). Continuous Formal Verification of Amazon s2n. In *Computer Aided Verification*, pages 430–446. Springer.

[Cohn-Gordon et al. 2016] Cohn-Gordon, K., Cremers, C., and Garratt, L. (2016). On post-compromise security. In *IEEE 29th CSF*, pages 164–178. IEEE.

[Cohn-Gordon et al. 2018] Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, J., and Milner, K. (2018). On ends-to-ends encryption: Asynchronous group messaging with strong security guarantees. In *ACM SIGSAC CCS*, pages 1802–1819.

[Cremers and Feltz 2012] Cremers, C. and Feltz, M. (2012). Beyond eck: Perfect forward secrecy under actor compromise and ephemeral-key reveal. In *ESORICS*, pages 734–751. Springer.

[Cremers 2006] Cremers, C. J. F. (2006). *Scyther: Semantics and verification of security protocols*. Eindhoven University of Technology Eindhoven.

[Dierks and Rescorla 2008] Dierks, T. and Rescorla, E. (2008). The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor. http://bit.do/rfc5246.

[Escarrone et al. 2019] Escarrone, T., Kreutz, D., and Fiorenza, M. (2019). Uma Primeira Analise do Ecosistema HTTPS no Brasil. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*. http://bit.do/wrseg19-uma.

[Hoffman 2005] Hoffman, P. (2005). Algorithms for Internet Key Exchange version 1 (IKEv1). RFC 4109, RFC Editor. https://tools.ietf.org/html/rfc4109.

[Homsirikamol et al. 2012] Homsirikamol, E., Morawiecki, P., Rogawski, M., and Srebrny, M. (2012). Security margin evaluation of SHA-3 contest finalists through SAT-based attacks. In *IFIP Int. Conf. on Comp. Inf. Sys. and Ind. Man.*, pages 56–67.

[Jenuario et al. 2019] Jenuario, T., Chervinski, J. O., Paz, G., and Kreutz, D. (2019). Verificação Automática de Protocolos de Segurança com a ferramenta Scyther. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*. http://bit.do/wrseg-scyther.

[Kim et al. 2006] Kim, B. H., Koo, J. H., and Lee, D. H. (2006). Robust e-mail protocols with perfect forward secrecy. *IEEE Communications Letters*, 10(6):510–512.

[Kivinen and Kojo 2003] Kivinen, T. and Kojo, M. (2003). More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). RFC 3526, RFC Editor. http://www.rfc-editor.org/rfc/rfc3526.txt.

[Krawczyk et al. 1997] Krawczyk, H., Bellare, M., and Canetti, R. (1997). HMAC: Keyed-Hashing for Message Authentication. RFC 2104, RFC Editor. http://www.rfc-editor.org/rfc/rfc2104.txt.

[Krawczyk and Eronen 2010] Krawczyk, H. and Eronen, P. (2010). HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869, RFC Editor. http://www.rfc-editor.org/rfc/rfc5869.txt.

[Kreutz et al. 2017] Kreutz, D., Yu, J., Esteves-Verissimo, P., Magalhaes, C., and Ramos, F. M. V. (2017). The KISS principle in Software-Defined Networking: An architecture for Keeping It Simple and Secure. *ArXiv e-prints*. https://arxiv.org/abs/1702.04294.

[Kreutz et al. 2018] Kreutz, D., Yu, J., Esteves-Veríssimo, P., Magalhães, C., and Ramos, F. M. V. (2018). The kiss principle in software-defined networking: A framework for secure communications. *IEEE Security Privacy*, 16(5):60–70.

[Kreutz et al. 2017] Kreutz, D., Yu, J., Ramos, F., and Esteves-Verissimo, P. (2017). ANCHOR: logically-centralized security for Software-Defined Networks. *ArXiv e-prints*. https://arxiv.org/abs/1711.03636.

[Kreutz et al. 2019] Kreutz, D., Yu, J., Ramos, F. M. V., and Esteves-Verissimo, P. (2019). ANCHOR: Logically centralized security for software-defined networks. *ACM Trans. Priv. Secur.*, 22(2):8:1–8:36.

[Krombholz et al. 2015] Krombholz, K., Hobel, H., Huber, M., and Weippl, E. (2015). Advanced social engineering attacks. *J. of Inf. Sec. and applications*, 22:113–122.

[Lehmann and Tackmann 2018] Lehmann, A. and Tackmann, B. (2018). Updatable encryption with post-compromise security. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 685–716. Springer.

[Leskin 2018] Leskin, P. (2018). The 21 scariest data breaches of 2018. Último acesso em: 2019-04-05. https://bit.ly/2uSLjVb.

[Li et al. 2018] Li, L., Sun, J., Liu, Y., Sun, M., and Dong, J. (2018). "a formal specification and verification framework for timed security protocols". *IEEE Trans. on Soft. Engineering*, 44(8):725–746.

[Ling et al. 2013] Ling, Z., Fu, X., Jia, W., Yu, W., Xuan, D., and Luo, J. (2013). Novel packet size-based covert channel attacks against anonymizer. *IEEE Transactions on Computers*, 62(12):2411–2426.

[Machado et al. 2019] Machado, R., Kreutz, D., Paz, G., and Rodrigues, G. (2019). Vazamentos de Dados: Histórico, Impacto Socioeconômico e as Novas Leis de Proteção de Dados. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*. http://bit.do/dleaks.

[Meyer et al. 2014] Meyer, C., Somorovsky, J., Weiss, E., Schwenk, J., Schinzel, S., and Tews, E. (2014). Revisiting SSL/TLS implementations: New bleichenbacher side channels and attacks. In *23rd USENIX Security Symposium*, pages 733–748.

[Poettering and Rösler 2018] Poettering, B. and Rösler, P. (2018). Asynchronous ratcheted key exchange. Cryptology ePrint Archive. http://bit.do/2018296.

[Rescorla 1999] Rescorla, E. (1999). Diffie-hellman key agreement method. RFC 2631, RFC Editor. http://www.rfc-editor.org/rfc/rfc2631.txt.

[Rescorla 2018] Rescorla, E. (2018). The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor.

[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.

[Rutishauser 2017] Rutishauser, D. (2017). About TLS Perfect Forward Secrecy and Session Resumption. http://bit.do/tls-pfs.

[Sun et al. 2005] Sun, H.-M., Hsieh, B.-T., and Hwang, H.-J. (2005). Secure e-mail protocols providing perfect forward secrecy. *IEEE Communications Letters*, 9(1):58–60.

[Thornburgh 2004] Thornburgh, T. (2004). Social engineering: the dark art. In *1st annual conf. on Information security curriculum development*, pages 133–135. ACM.

[Times 2018] Times, N. Y. (2018). Cybersecurity Firm Finds Way to Alter WhatsApp Messages. http://bit.do/nyt-wa.

[Yang and Shieh 1999] Yang, W.-H. and Shieh, S.-P. (1999). Password authentication schemes with smart cards. *Computers & Security*, 18(8):727 – 733.

[Yu et al. 2018] Yu, J., Ryan, M., and Cremers, C. (2018). DECIM: Detecting Endpoint Compromise In Messaging. *IEEE Trans. on Information Forensics and Security*, 13(1):106–118.

[Zenger et al. 2016] Zenger, C. T., Pietersz, M., and Paar, C. (2016). Preventing relay attacks and providing perfect forward secrecy using physec on 8-bit μc. In *IEEE ICC*, pages 110–115.

[Zhang et al. 2014] Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T. (2014). Cross-tenant side-channel attacks in paas clouds. In *ACM SIGSAC CCS*, pages 990–1003.

Capítulo 3

Introdução à linguagem de programação P4, o futuro das redes

Pedro Eduardo Camera e Alisson Borges Zanetti

Abstract

This short course aims to explain the functioning of the P4 language, demonstrating its relevance today through examples of practical application, such as monitoring links between network equipment. With a theoretical approach and exercises, this paper discusses the main points of SDN networking and how the programming of packets in the data plane, in the figure of P4, enables the increase of user experience, troubleshooting, better resources usage, and network control centralization.

Resumo

Este minicurso tem como objetivo explanar o funcionamento da linguagem P4, demostrando sua relevância nos dias atuais por meio de exemplos de aplicação prática, como o monitoramento e links entre os equipamentos de rede. Dispondo de um referencial teórico e exercícios, este documento discorre sobre os principais pontos das redes SDN e de como a programação de pacotes no plano de dados, na figura do P4, possibilita o aumento da experiência de usuário, capacidade de troubleshooting, otmização do uso de recursos e centralização do controle de rede.

3.1. Introdução

O monitoramento de rede é crucial na identificação e resolução de problemas, sendo implementado desde empresas de pequeno porte a operadoras de alta performance. Todavia, à medida que as redes se expandem, cresce a complexidade de gerenciamento e *troubleshooting* em suas estruturas. Outra adversidade é o destaque da experiência de usuário na formulação das aplicações, compondo-se insuficiente, frente às exigências atuais de desempenho, a mera disponibilidade e redundância dos serviços. A heterogeneidade também é um fator contribuinte, posto que os fabricantes, a fim de diferenciar seus produtos, empregam soluções proprietárias que dificultam o desenvolvimento de ferramentas de

monitoramento abrangentes e efetivas, impelindo o uso de utilitários reativos e datados (como *traceroute*, SNMP, ICMP, dentre outros) para tal tarefa.

Com o surgimento das redes definidas por software (*Software Defined Networking*), tratadas daqui para frente como SDN, novas formas de monitoramento tornaram-se possíveis. Apresentada em 2008, esse tipo de rede tem como principais características a separação das camadas de controle e dados, a centralização da gerência dos ativos e a visualização global do estado da rede [Singh and Jha 2017]. Esse panorama é alcançado por meio da definição de programas customizáveis que intermedeiam (e ditam) as ações do hardware. Nos últimos anos, o paradigma SDN se popularizou especialmente em redes que comportam aplicações de alta demanda [Haleplidis et al. 2015].

Apesar disso, com o passar do tempo, a centralização SDN deslocou a programabilidade ao baixo nível (camada de dados), a fim de balancear a carga imposta ao gerenciador SDN (camada de controle). A contar de 2014, o aparecimento de linguagens de processamento de pacotes, como o P4 (*Programming Protocol-Independent Packet Processors*), bem como seu suporte em hardware, possibilitaram ao plano de dados a execução de tarefas atreladas unicamente aos controles superiores. Com isso, uma nova gama de aplicações, incluindo o monitoramento de redes em tempo real, começaram a ser esboçadas.

O monitoramento tem como base a busca por informações de *status* da rede e sua demonstração aos administradores. O presente artigo propõe um exercício sobre essa nova tecnologia, a saber o P4, utilizando-se de um algoritmo nele programado para realizar varreduras acerca do estado de rede dos equipamentos. Por encontrar-se no baixo nível, implementações P4 conseguem capturar metadados restritos ao hardware. Com isso, seu nível de precisão e detalhamento aprofundam a compreensão das atividades de rede.

Este trabalho divide-se da seguinte forma: uma breve fundamentação teórica a respeito dos conceitos SDN (seção 3.2.1) e P4 (seção 3.2.2.1 em diante) são apresentados, seguidos de uma descrição das ferramentas essenciais à execução desta proposta (seção 3.3) e das particularidades da implementação do programa P4 em si (seção 3.4), encerrando-se com a conclusão e deposições finais (seção 3.5). Demais detalhes são providos pelas informações anexas a este minicurso (seção 3.6).

3.2. Referencial Teórico

3.2.1. SDN

O paradigma SDN foi desenvolvido por cientistas da Universidade de Stanford, sendo fruto de esforços de egressos de outras universidades americanas, como Berkeley, Carnigie Mellon e Princeton [Sood and Xiang 2017]. Segundo a ONF (*The Open Networking Foundation*) [1], o objetivo da SDN é reduzir custos e melhorar a experiência do usuário, automatizando toda a gama de serviços de rede. Seus princípios incluem o desacoplamento entre controle e dados, a capacidade de interação direta dos elementos da rede com este e a centralização do gerenciamento [OPEN NETWORK FOUNDATION 2016].

A capacidade de dissociar os planos dá-se ao tornar a camada de controle a respon-

[1] Consórcio sem fins lucrativos que desenvolve, padroniza e comercializa soluções SDN.

sável pela decisão de roteamento, analisando os pacotes e resolvendo a melhor forma de lidar com o tráfego, enquanto a camada de dados apenas encaminha de acordo com a deliberação tomada [Singh and Jha 2017]. Através de fluxos, que qualificam uma sequência de movimentos entre origem e destino, as determinações de envio são formuladas, sendo os pacotes balizados por critérios de correspondência e ação (*match-action*). Nos equipamentos, pacotes de mesmo fluxo recebem políticas de serviço idênticas. Essa abstração uniformiza o comportamento dentro dos diferentes dispositivos de rede, tais quais roteadores, *switches*, *firewalls* e *middleboxes* [Kreutz et al. 2015].

O aprovisionamento de um plano de controle unificado, onde um único software gere diversos planos de dados com múltiplos membros, ocorre por meio de protocolos como o OpenFlow [Singh and Jha 2017]. A Figura 3.1 representa todas essas vertentes, contrastando-as com a arquitetura tradicional. Percebe-se a forma consolidada com que as relações SDN ocorrem, onde tanto as aplicações de alto nível quanto as operações de baixo têm a supervisão do controlador.

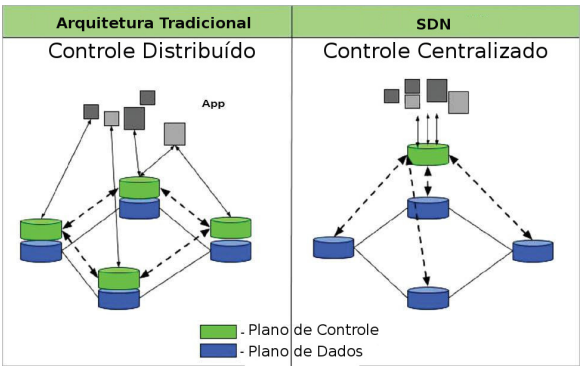


Figura 3.1: Diferenças entre rede tradicional e SDN. Fonte: Autores.

3.2.1.1. OpenFlow

Conforme mencionando, a centralização do controle é obtida através do uso do protocolo aberto OpenFlow [Huang et al. 2016], responsável pelo encorajamento dos fornecedores na implementação de alguns conceitos SDN em produtos de rede [Kreutz et al. 2015]. Proposto por McKeown et al. [McKeown et al. 2008] [Sood and Xiang 2017], o protocolo é utilizado na maioria das propostas SDN, possuindo diversas aplicações *open source* de controladores. Experimentos iniciais utilizando-se OpenFlow foram criados para separar a rede em uma fatia de software controlável, com foco no encaminhamento de pacotes. O protocolo aproveita-se do fato de *switches* e roteadores modernos possuírem tabelas de fluxo para funções de roteamento, máscaras de sub-rede, proteção de *firewall* e análise estatística do fluxo de dados [Xia et al. 2015].

3.2.1.2. Comutação SDN

Dependendo das regras instaladas pelo aplicativo controlador, um *switch* OpenFlow pode se comportar como um roteador, comutador, *firewall*, *load balancer*, *traffic shaper* ou quaisquer outras funções que caracterizam um *middlebox* de rede [Kreutz et al. 2015]. A Figura 3.2 traz de modo simplificado o funcionamento da comutação SDN. Os *switches* OpenFlow suportam a operação de tabelas de fluxo, por onde o protocolo OpenFlow combina e processa pacotes de rede pelas regras nelas estabelecidas. Cada entrada da tabela constitui-se de três componentes (cabeçalho, instrução e estatísticas) em vez da entrada de roteamento tradicional (quíntupla IP). Os pacotes são correspondidos (*matching*) por seus campos de cabeçalho (*header*) e, em seguida, processados de acordo com a instrução (*action*) no fluxo de entrada. As estatísticas indicam o *status* da rede, incluindo prioridade, contadores, e assim por diante [Gong et al. 2015].

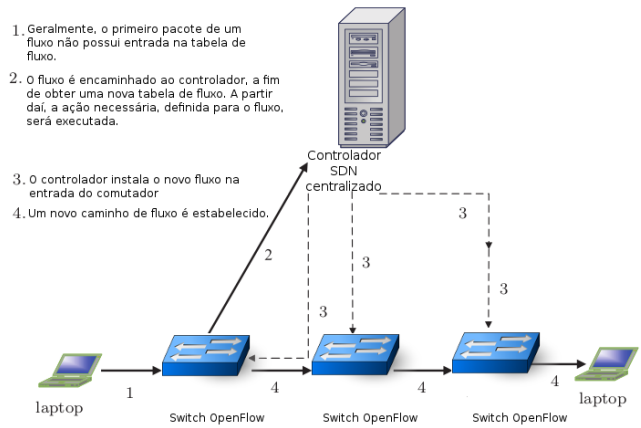


Figura 3.2: Comutação com Switch OpenFlow. Fonte: SOOD, K; XIANG, Y. [Sood and Xiang 2017].

3.2.2. Programabilidade no plano de dados

Consoante o já exposto, as camadas de controle e dados caracterizam-se como fisicamente separadas [Santiago da Silva et al. 2018] [Bosshart et al. 2014], o que permite a evolução de ambas distintamente. Essa separação permitiu que os dispositivos de encaminhamento atingissem níveis maiores de programabilidade. Não obstante essa capacidade, houve um enredamento destes ao plano de controle. Ao utilizar-se de uma interface comum, aberta e independente de provedor (OpenFlow), a camada de controle revelou-se imprescindível ao plano de dados [Bosshart et al. 2014].

Inicialmente, a simplicidade satisfez a demanda. Porém, para permitir que dispositivos vigentes exponham mais de seus recursos ao controlador, a especificação OpenFlow (diretiva do uso do protocolo) tem-se tornado cada vez mais complexa, acrescendo campos e estágios de regras [Bosshart et al. 2014]. Embora continuamente atualizada, a catalogação fica aquém da proliferação de novos campos de cabeçalho. Em outras palavras, a padronização da programabilidade no plano de controle produziu um plano de

dados “fixo”, no sentido de que os usuários restringiram-se a trabalhar com protocolos identificados na especificação OpenFlow [Hancock and van der Merwe 2016].

3.2.2.1. P4

Para contornar as deficiências do OpenFlow [Santiago da Silva et al. 2018], o emprego de linguagens de programação no plano de dados vem ganhando adoção tanto na academia quanto na indústria. Abordagens como o P4 permitem descrever um comportamento agnóstico de encaminhamento no plano de dados. Por ser uma linguagem de alto nível, o P4 fornece um dialeto de rede simples para descrever o caminho dos datagramas, podendo trabalhar em conjunto com protocolos de controle SDN [Bosshart et al. 2014]. Com a proposta do P4, busca-se a evolução do modelo por meio da:

- **Reconfigurabilidade no campo:** os programadores devem ser capazes de mudar a forma como os *switches* processam os pacotes depois de implementados;
- **Independência do protocolo:** os comutadores não devem estar vinculados a nenhum protocolo de rede específico;
- **Independência do alvo:** os programadores devem ser capazes de descrever a funcionalidade de processamento de pacotes independentemente das especificidades do hardware subjacente;

3.2.2.2. PISA

Baseado na arquitetura PISA (*Protocol-Independent Switch Architecture*), compõe-se um programa P4 por declarações de cabeçalho (*header*), máquina de estado do analisador de pacotes (*parser*) e das tabelas de ação de correspondência (*match-action*). A linguagem P4 assume a implementação de um *parser* que percorre os *headers* do início ao fim, extraindo os valores de campo conforme o percurso. As medidas extraídas são enviadas para o processamento das tabelas de *match-action* [Bosshart et al. 2014], similar ao tratamento adotado pelo OpenFlow a nível de controle. Assim, é descrita a máquina de estado como o conjunto de transições de um cabeçalho para o próximo. A Figura 3.3 demonstra um caminho hipotético rastreado pelo *parser*, conforme a sequência de cabeçalhos extraídos.

As declarações de *header* especificam os nomes e as larguras dos campos para os cabeçalhos de protocolo nos quais o programa P4 se destina a operar. Em geral, cada cabeçalho contém uma lista ordenada de nomeações e seus respectivos tamanhos. A tabela *match-action* desempenha a principal função, identificando os campos de pacotes e metadados a serem lidos, além das possíveis ações executadas em resposta. As *actions* são funções parametrizáveis que invocam uma ou mais primitivas de linguagem [2]. Uma vez que as tabelas e ações são definidas, resta especificar o fluxo de controle entre uma tabela e outra, por meio de um conjunto de funções, condicionais e referências de tabela [Bosshart et al. 2014] [Hancock and van der Merwe 2016].

[2]Elementos mais simples existentes numa linguagem de programação.

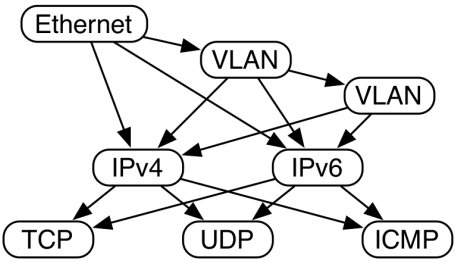


Figura 3.3: Representação do *parser* numa rede hipotética. Fonte: GIBB, G. et al. [Gibb et al. 2013].

A Figura 3.4 traz a abstração presente no envio de dados dentro do modelo P4/PISA. Primeiramente, os pacotes são manipulados pelo *parser*. Este reconhece e retira campos do *header* e, assim, define os protocolos suportados pelo comutador. Os campos extraídos são passados para as tabelas de *match-action*, divididas entre entrada (*ingress*) e saída (*egress*). Embora ambas possam modificar o *header*, a *ingress match-action* determina a(s) porta(s) de *egress* e a fila na qual o pacote é colocado. Com base nesse processamento, o pacote pode ser encaminhado, replicado (para *multicast*, *span* ou até ao plano de controle), descartado ou mesmo acionar o controle de fluxo. A *egress match-action* executa modificações por instância no *header*, por exemplo, as cópias *multicast*. As tabelas de ação (contadores, *policers* e assim por diante) podem ser associadas a um fluxo para rastrear o estado *frame-to-frame*, bem como informações de metadados que ofertam detalhes entre os estágios percorridos.

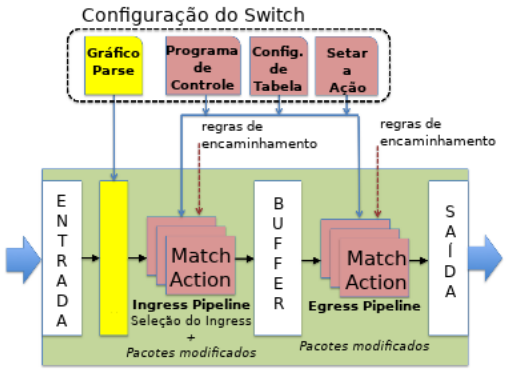


Figura 3.4: Abstração do modelo de encaminhamento. Fonte: BOSSHART, P. et al. [Bosshart et al. 2014].

3.2.2.3. Compilador P4

Com a popularização, diversos compiladores P4 emergiram recentemente. Neste trabalho, atentaremos ao P4C (compilador de referência), onde o processo de compilação é definido

em dois períodos. Num primeiro momento, o programa de controle P4 é convertido para uma representação gráfica das dependências entre as tabelas, conhecida como TDG (*Table Dependency Graphs*). Essa dependência estipula quais tabelas podem ser executadas em paralelo. Em analogia, podemos realizar a leitura simultânea dos conteúdos da tabela IP e ARP, pois ambas possuem dependência de dados entre si. Posteriormente, essa exibição é utilizada por um *back-end* específico de destino, que consome os dados na construção dos recursos especificados para o dispositivo [Bosshart et al. 2014]. Para uma melhor compreensão, observa-se o esquema expresso na Figura 3.5, que traz um exemplo de TDG para um contexto de *switches* L2/L3. Os nós TDG são mapeados diretamente para as tabelas de *match-action*, na qual uma análise de dependência identifica a colocação de cada tabela dentro do *pipeline*.

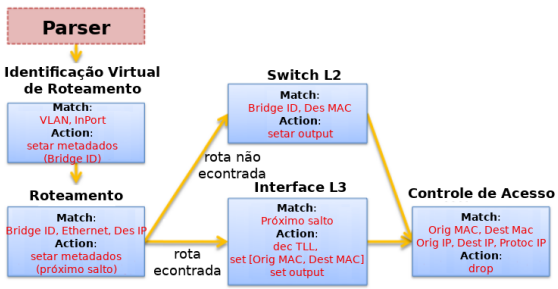


Figura 3.5: Gráfico de Dependência de Tabela (TDG) para um dispostivo L2/L3. Fonte: Adaptado de BOSSHART, P. et al. [Bosshart et al. 2014].

Corroborando o processo acima, a Figura 3.6 repassa as etapas de *parsing* do código-fonte P4 para então produzir uma representação intermediária (IR) de alto nível. Um *back-end* de compilador converte esta IR numa conformação específica de destino (por exemplo, código binário ou JSON). Como ofertas de *back-end* do compilador P4, podemos incluir o próprio P4C [Hancock and van der Merwe 2016]. Em dispositivos com *parsers* programáveis, o compilador converte a descrição em máquina de estado, enquanto nos fixos apenas verifica a descrição quanto à consistência com o alvo (*target*) pretendido [Bosshart et al. 2014]. Ou seja, a máquina de estado identifica, sequencialmente, a ordem e as relações do cabeçalho dentro do pacote. Começando do nó raiz, as transições de estado são tomadas em resposta aos próximos valores de campo de cabeçalho [Gibb et al. 2013].

3.3. Implementação

3.3.1. Mininet, BMv2 e P4Runtime

Antecipadamente, para o bom andamento do experimento, é necessário principiar o cursista a algumas noções básicas sobre o ambiente de testes. Para emular o âmbito SDN/P4, utiliza-se a ferramenta Mininet, um sistema computacional que imita o funcionamento de *switches*, controladores e *hosts* em tempo real. Neste sentido, o Mininet é um dos aplicativos mais empregados para avaliação de redes SDN, permitindo a emulação a partir de uma única máquina. Aproveitando-se de premissas de virtualização, este utilitário per-

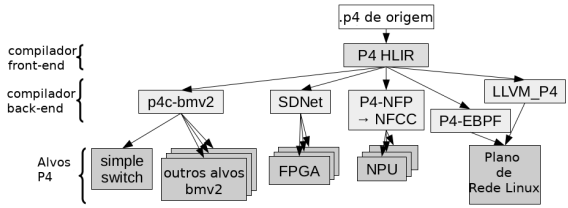


Figura 3.6: Compilando um programa P4 para vários alvos. Fonte: HANCOCK, D.; MERWE, J. van der. [Hancock and van der Merwe 2016].

mite ao usuário criar, interagir, personalizar e compartilhar um protótipo de rede completo que integra-se a controladores reais e demais atores externos.

Como a emulação tem suas limitações, é preciso estabelecer um padrão de procedimento fidedigno a uma rede P4 real. Nesta situação, um simulador de redes reproduz o comportamento de equipamentos reais através de um modelo matemático. Somando-se, o Mininet suporta o plano de dados programável com a ajuda do BMv2 (*Behavioral Model version 2*), que dita o procedimento dos *switches* P4 dentro da estrutura emulada. Outro agente a ser considerado é o *framework* P4Runtime que, nesta aplicação, desempenha as funções do plano de controle, gerenciando os elementos do plano de dados definidos em P4. A Figura 3.7 engloba todos os itens citados e suas correspondentes vinculações.

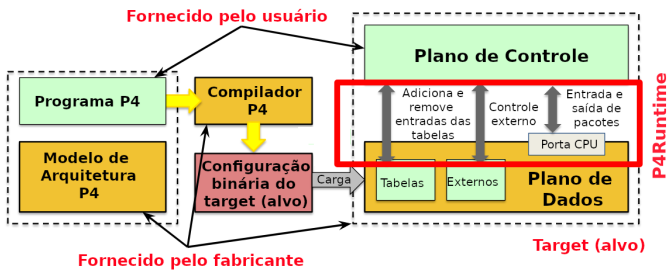


Figura 3.7: Fluxo de implementação do programa P4. Fonte: Autores.

3.3.2. Máquina Virtual

Visando o benefício máximo do tempo e a organização, dispõe-se uma máquina virtual contendo pré-instalações de pacotes de sistema e softwares que serão manipulados durante o laboratório. A mesma pode ser obtida através deste *hyperlink*[3]. Para se abrir o arquivo, recomenda-se a instalação do programa VirtualBox em sua versão 6.0.10 (ou superior) na máquina hospedeira.

[3]https://drive.google.com/a/upf.br/file/d/1E9KXKYV6cbyO-X-qR_5FCtaiLjVONkn0/view?usp=sharing

3.4. Programa P4 para monitorar *links*

Nesta seção apresenta-se uma prática P4 que objetiva um programa de monitoramento de ativos no plano de dados, por meio da coleta de informações sobre a rede, em especial do estado do *link*, seguindo a topologia exposta na Figura 3.8. Nela, ilustra-se a estrutura criada no emulador Mininet, na qual a simulação de comportamento dos comutadores fica a cargo do BMv2. Dispondo de quatro *switches*, nota-se que os superiores (Switch3 e Switch4) interligam-se com os da base (Switch1 e Switch2) e que estes, por sua vez, comunicam-se com os *hosts* 1, 2, 3 e 4, respectivamente. Para facilitar o entendimento, os *links* são também enumerados, posto que *switches* e *hosts* conectam-se por vários canais, quase formando uma *mesh*.

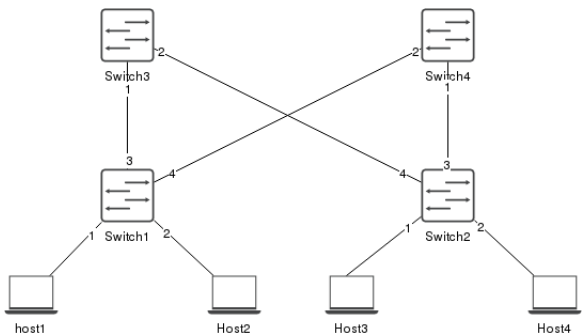


Figura 3.8: Topologia de rede básica do minicurso. Fonte: Autores.

Para que o objetivo de monitoramento seja alcançado, o *host*1 encaminha um pacote que coleta metadados referentes ao estado dos *links* nos *switches*. Por este motivo, o pacote tem de trafegar por todos caminhos entre os comutadores, retornando para a sua origem ao final. A Figura 3.9 descreve como o pacote "sonda" realiza a captura da quantidade de *bytes* da porta de saída (*egress*) de cada equipamento. Acompanhando o caminho traçado na imagem, vê-se que a "sonda" percorre os *switches* 1, 4, 2 e 3 na primeira volta (trajeto em vermelho), retornando por 1, 3, 2, 4 e, subsequentemente, 1 novamente (destacado em azul).

Inicialmente, os cabeçalhos foram estruturados com base no que é apresentado na Figura 3.10. Foram definidos os *headers* padrão Ethernet e IPv4. Para o perfeito funcionamento da aplicação, o último tem seu tráfego verificado com base na tabela de rota LPM (*Longest Prefix Match*) [4]. Acima, formulou-se novos cabeçalhos, como o *probe*, que conta a quantidade de saltos (*hops*), e o *probe_data*, onde armazenam-se as informações coletadas em forma de pilha. Por último, temos o *probe_fwd*, que conserva a informação da interface de saída (*egress*) precedente de cada *hop* em que o pacote transitou.

Conforme explicado na seção 3.2.2.3, os programas P4 observam uma ordem de operações. A primeira delas é o *parser*, que realizará a extração dos dados dos pacotes, seguido do processamento de *ingress* e *egress* e, posteriormente, o *deparser*, passo em que os cabeçalhos são remontados. Como visto, de cada estado do *parser* deriva-se um tipo de informação diferente. No nosso programa, o analisador inicia pelo estado de *start*

[4] Algoritmo que procura o prefixo IP de destino do próximo *hop*.

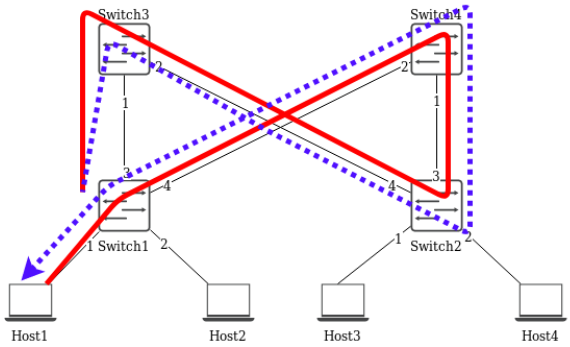


Figura 3.9: Caminho do pacote “sonda”. Fonte: Autores.

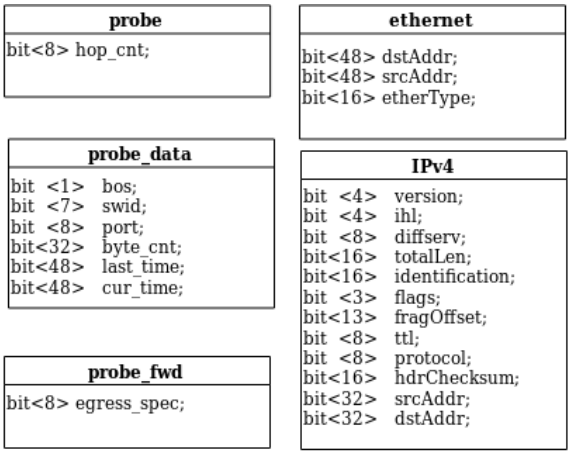


Figura 3.10: Cabeçalhos definidos neste exemplo. Fonte: Autores.

que, uma vez invocado, destina-se ao próximo estágio (que efetuará a retirada Ethernet). Conforme a Figura 3.11, percebe-se que este é um dos pontos críticos do procedimento. Caso o tipo aferido no pacote corresponder a 0x800, trata-se de um pacote IPv4 ordinário que, após ter sua informação de *header* coletada, é direcionado para o processamento de entrada (*ingress*) imediatamente. Se o tipo for 0x812, significa que é um cabeçalho *probe* (e consequentemente um pacote “sonda”), devendo deslocar-se por mais verificações. Do contrário, nenhum *header* se deduz nesta etapa, passando restritamente ao *ingress*.

Ainda na Figura 3.11, para coletar dados do *header probe* inicia-se apurando a quantidade de *hops* por onde o pacote passou. Se esse número for igual a 0, concerne-se que o pacote ainda não passou por nenhum *switch*, continuando assim para o estado de encaminhamento. Caso contrário, assume-se que o pacote adentrou previamente nas estruturas de rede, partindo para a extração do *probe_data* até que a coleta da pilha LIFO (*Last In, First Out*) chegue ao valor 1, o que equivale a leitura completa da fila, destinando-se também para o estado de encaminhamento. Neste último, é realizada a coleta do *probe_fwd*

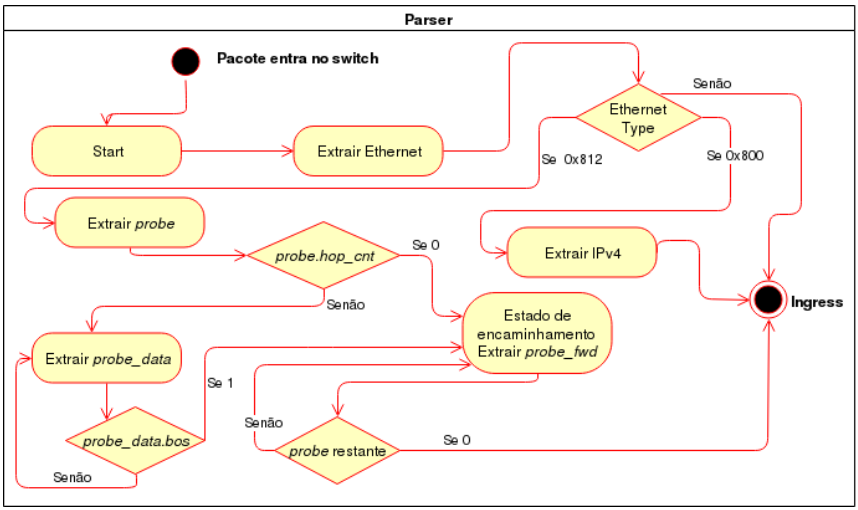


Figura 3.11: *Parser* do programa de monitoramento de *links*. Fonte: Autores.

e, ulteriormente, enviado ao *ingress*.

A execução do *ingress* caracteriza-se como o momento de entrada dos pacotes nos *switches*. Se o pacote for IPv4, suas informações de cabeçalho são transferidas para a tabela *match-action* correspondente (neste caso IPv4). Em caso de primeira execução de fluxo, a tabela consulta o P4Runtime (que orienta-se por uma tabela LPM) a fim de descobrir quais ações serão consumadas. Ilustradas na Figura 3.12, as ações condizentes apresentam dois caminhos possíveis: o encaminhamento do pacote para a porta de saída (*egress*), sofrendo incrementação do *ttl* (tempo de vida) e substituição de MAC ou então o descarte deste. Doutra feita, se o pacote contiver a função de coleta de dados do *switch* (“sonda”), empreendem-se a expedição do pacote para a porta de saída (*egress*) e incrementação da quantidade de equipamentos no cabeçalho *probe*.

A fase *egress*, pormenorizada na Figura 3.13, pode ser encarada como a mais crucial deste programa de monitoramento. Nela, além de recolhidas as informações de estado do *link*, efetivam-se os cálculos quanto a sua utilização. Para isso, logo que o pacote adentra à etapa, busca-se determinar o *timestamp* [5] atual do *egress*, assim como recuperar o da passagem anterior, com o propósito de avaliar qual foi o intervalo de tempo desde a última coleta. Um dos componentes mais importantes para a aferição do uso do *link* é o registro *byte_cnt_reg*, que se ocupa da quantidade de *bytes* trafegados, sendo persistido no plano de controle (P4Runtime). Através de uma *action* (ação P4), pode-se acessá-lo no plano de dados. Neste caso, se já houver informação de *bytes*, requesta-se o valor antigo e soma-se ao atual, modificando a variável local *byte_cnt*. Na hipótese de ser uma nova entrada, ocorre somente a escrita por meio de outra variável (*new_byte_cnt*).

Atendo-se novamente à Figura 3.13, o programa avalia se o cabeçalho é uma *probe*. Em caso positivo, o plano de dados envia uma *action* para o controle (P4Runtime) resetar o registro de *bytes* contido nele. Em oposto (ou após o *reset*), analisa-se uma vez

[5] Cadeia de caracteres que denota hora ou data de certo evento.

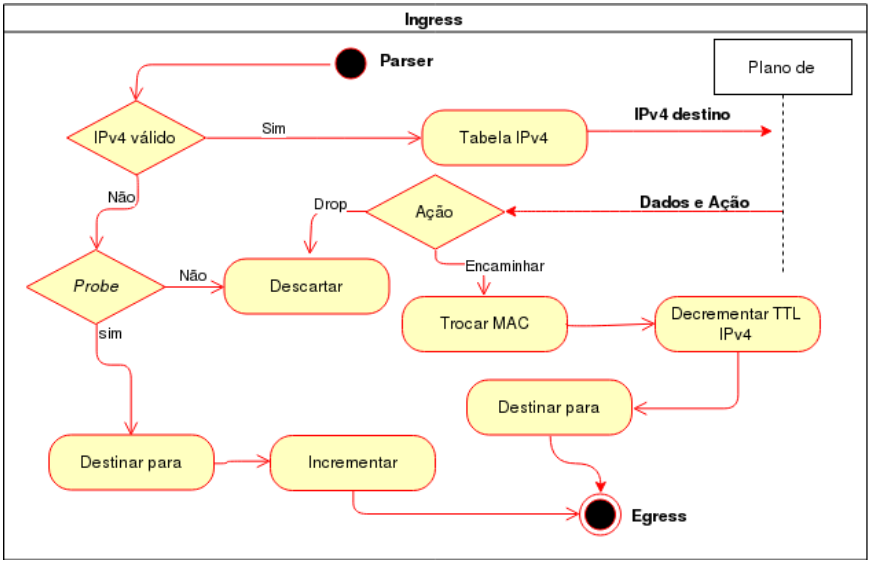


Figura 3.12: Esquema do funcionamento do *ingress* na aplicação. Fonte: Autores.

mais se o pacote é uma "sonda". Se a condição for invalidada reiteradamente, o *switch* encaminha o pacote ao *deparser*. Do contrário, tem-se a certeza que é uma *probe* válida, perfazendo-se os próximos passos. Sucessivamente, é ativado o *header* para armazenar os dados sobre a pilha (*probe_data*). Na circunstância deste ser o primeiro da fila, ativa-se o *bit* do campo *bos*, um controle que possibilitará a leitura dos metadados no *parser* sem *looping*. Ao final, há a condução para a tabela que solicita o ID do *switch* ao P4Runtime, salvando-o na estrutura.

Subsequentemente, adiciona-se o ID do *egress*, os *bytes* contabilizados no *byte _cnt_reg* e o último *timestamp* do pacote final trafegado no *switch*, bem como o *timestamp* atual. Nesta etapa, também é realizada a atualização de registro do último *timestamp* no plano de controle. Finalizando, o pacote é redirecionado ao *deparser*, que realiza a montagem dos cabeçalhos dentro do pacote. Desta forma, todas as atualizações dos *headers* anteriores desempenhadas sobre estrutura na memória do plano de dados são alocados no pacote que irá afluir pelos *switches*.

Devido à lógica do programa estar plenamente baseada sobre o processamento do *ingress* e *egress*, em anexo está a etapa do processo de entrada e saída, nas seções A.1, A.2, na devida ordem. Nota-se que os algoritmos apresentados estão de absoluto acordo com os diagramas retratados nesta seção.

3.5. Execução do projeto

Similar ao exposto na seção 3.3.2, para realizar a programação no interior do plano de dados, disponibiliza-se uma máquina virtual com os softwares necessários. Também foi criado um repositório GitHub, acessível pelo *hyperlink*[6], onde os arquivos básicos para a

[6]https://github.com/PedroEduardo68/without-code

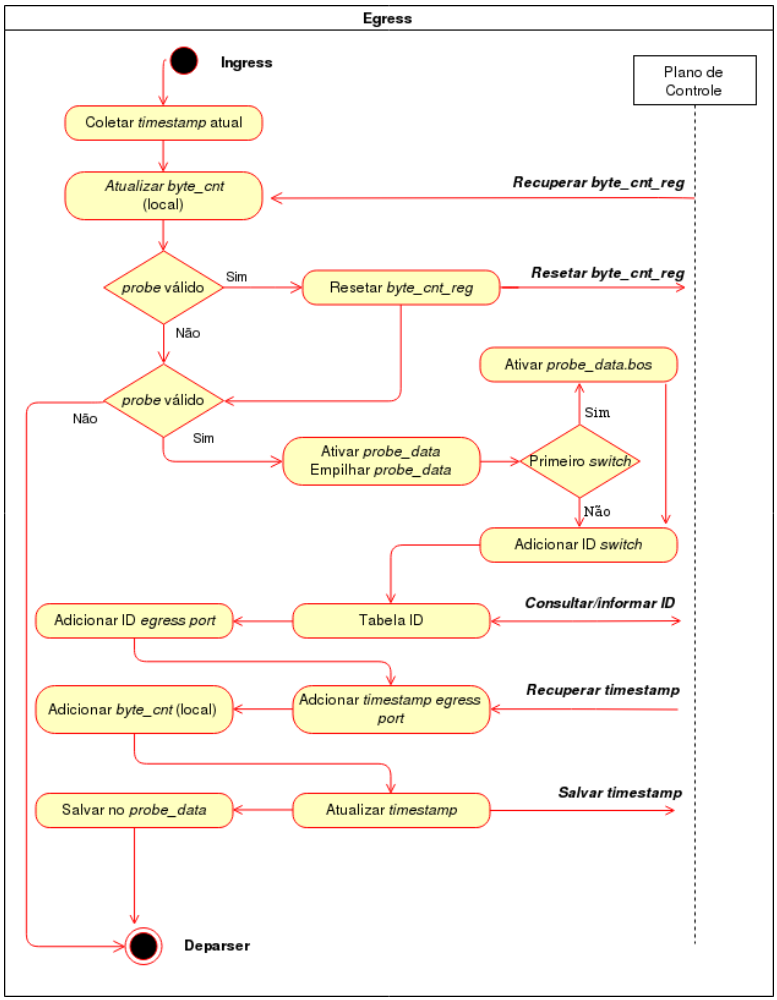


Figura 3.13: Fluxograma do *egress* adotado. Fonte: Autores.

compilação dos códigos encontram-se providenciados. Abaixo, estão descritas as etapas a serem seguidas para execução do código e do ambiente:

- **Clonar arquivos-** Primeiramente, deve-se clonar o repositório GitHub para dentro do diretório da máquina virtual, preferencialmente em `/home/p4`:

```
$cd /home/p4/
$git clone https://github.com/PedroEduardo68/without-code
```

- **Adicionar diretório-** Com o objetivo organizar os arquivos para compilação, copiar o diretório `exercises` dentro da pasta `/home/p4/tutorials/`:

```
$mv exercises /home/p4/tutorials/
```

- **Codificar-** Para codificar, conforme o detalhamento presente na seção 3.4, deve-se aceder ao diretório `/home/p4/tutorials/exercises/link_monitor`, sendo obrigatório o desenvolvimento do código dentro do arquivo `link_monitor.p4`.

Com o programa concluído, parte-se para a avaliação do estado da rede, verificando-se a transmissão do tráfego de dados pelas interfaces por meio do software P4. Ao acessar a pasta `link_monitor`, executar o comando:

```
$make run
```

O comando acima procede com a montagem da topologia de rede (*switches*, *hosts* e *links*) ao mesmo tempo que compila o arquivo `link_monitor.p4`, já incorporado dentro dos alvos de rede. Subsequentemente, abre-se o *shell* do Mininet, possibilitando operações sobre os equipamentos. Para entrar no console do Host1 duas vezes (para operar os arquivos `receive.py` e `send.py`, separadamente), usa-se:

```
$xterm h1 h1
```

O programa `receive.py` "escuta"a interface em que os pacotes com dados provenientes dos *switches* ingressam. O `send.py` os encaminha. Retornando ao *shell* do Mininet, executa-se o software `iperf` para gerar tráfego entre os computadores finais, por meio do comando:

```
$iperf h1 h4
```

Durante a execução do aplicativo gerador de trânsito, ao conectar no *prompt* do Host1 (que recebe os dados), nota-se que as interfaces estão demostrando o valor real de circulação. Com o projeto operando, para finalizar o ambiente de *testbed* Mininet, utilizam-se os comandos:

```
$quit
$make stop
$make clean
```

3.6. Conclusão

Conclui-se, pelos exemplos aqui demonstrados, a potencialidade da programabilidade no plano de dados por meio da linguagem P4. Alinhada aos conceitos SDN neste abarcados, assevera-se sua importância na transformação das redes de computadores, viabilizando o provisionamento de demandas atuais e futuras. Ao expressar os dispositivos independentemente de protocolos, o paradigma P4 quebra o enredamento imposto pelo plano de controle, até então o único elemento coordenável, resolvendo diversas dificuldades na programabilidade direta dos ativos de rede.

Buscou-se nas explanações contidas neste artigo denotar o funcionamento básico dos principais componentes P4, familiarizando o cursista às nomenclaturas e o *modus operandi* da linguagem, através da exposição de esquemas e figuras que simplificam o modelo. Com uso prático, os exemplos são facilmente assimilados, proporcionando uma rápida curva de aprendizagem e fomentando o interesse neste tipo de tecnologia. Logo, espera-se que este trabalho seja um instrumento válido de difusão do conhecimento acerca do SDN e P4 e, conseguintemente, contribua no crescimento de produções vindouras.

A. Anexos

A.1. Código do processamento de entrada

```
/*************************************
*  I N G R E S S   P R O C E S S I N G
*************************************/

action drop() {
    mark_to_drop(standard_metadata);
}

action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    standard_metadata.egress_spec = port;
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}

table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = drop();
}

apply {
    if (hdr.ipv4.isValid()) {
        ipv4_lpm.apply();
    }
    else if (hdr.probe.isValid()) {
        standard_metadata.egress_spec = (bit<9>)meta.egress_spec;
        hdr.probe.hop_cnt = hdr.probe.hop_cnt + 1;
    }
}
```

A.2. Código do processamento de saída

```
/*************************************
* E G R E S S   P R O C E S S I N G
*************************************/

register<bit<32>>(MAX_PORTS) byte_cnt_reg;
register<time_t>(MAX_PORTS) last_time_reg;

action set_swid(bit<7> swid) {
    hdr.probe_data[0].swid = swid;
}
table swid {
    actions = {
        set_swid;
        NoAction;
    }
    default_action = NoAction();
}

apply {
    bit<32> byte_cnt;
    bit<32> new_byte_cnt;
    time_t last_time;
    time_t cur_time = standard_metadata.egress_global_timestamp;
    byte_cnt_reg.read(byte_cnt, (bit<32>)standard_metadata.egress_port);
    byte_cnt = byte_cnt + standard_metadata.packet_length;
    new_byte_cnt = (hdr.probe.isValid()) ? 0 : byte_cnt;
    byte_cnt_reg.write((bit<32>)standard_metadata.egress_port, new_byte_cnt);

    if (hdr.probe.isValid()) {
        hdr.probe_data.push_front(1);
        hdr.probe_data[0].setValid();

        if (hdr.probe.hop_cnt == 1) {
            hdr.probe_data[0].bos = 1;
        }
        else {
            hdr.probe_data[0].bos = 0;
        }
        swid.apply();
        hdr.probe_data[0].port = (bit<8>)standard_metadata.egress_port;
        hdr.probe_data[0].byte_cnt = byte_cnt;
        last_time_reg.read(last_time, (bit<32>)standard_metadata.egress_port);
        last_time_reg.write((bit<32>)standard_metadata.egress_port, cur_time);

        hdr.probe_data[0].last_time = last_time;
        hdr.probe_data[0].cur_time = cur_time;
    }
```

}

Referências

[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.

[Gibb et al. 2013] Gibb, G., Varghese, G., Horowitz, M., and McKeown, N. (2013). Design principles for packet parsers. In *Architectures for Networking and Communications Systems*, pages 13–24, San Jose, CA, USA. IEEE.

[Gong et al. 2015] Gong, Y., Huang, W., Wang, W., and Lei, Y. (2015). A survey on software defined networking and its applications. *Frontiers of Computer Science*, 9(6):827–845.

[Haleplidis et al. 2015] Haleplidis, E., Hadi Salim, J., Denazis, S., and Koufopavlou, O. (2015). Towards a network abstraction model for sdn. *Journal of Network and Systems Management*, 23(2):309–327.

[Hancock and van der Merwe 2016] Hancock, D. and van der Merwe, J. (2016). Hyper4: Using p4 to virtualize the programmable data plane. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 35–49, New York, NY, USA. ACM.

[Huang et al. 2016] Huang, T., Yu, F. R., and Liu, Y.-j. (2016). Special issue on future network: Software-defined networking. *Frontiers of Information Technology & Electronic Engineering*, 17(7):603–605.

[Kreutz et al. 2015] Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. In *Proceedings of the IEEE*, volume 105, pages 14–76, Singapore. IEEE Computer Society.

[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. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

[OPEN NETWORK FOUNDATION 2016] OPEN NETWORK FOUNDATION (2016). Sdn architecture - a primer. Disponível em: <https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2013/05/7-26%20SDN%20Arch%20Glossy.pdf. Acessado: 19-08-2019.

[Santiago da Silva et al. 2018] Santiago da Silva, J., Boyer, F.-R., and Langlois, J. P. (2018). P4-compatible high-level synthesis of low latency 100 gb/s streaming packet parsers in fpgas. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, pages 147–152, New York, NY, USA. ACM.

[Singh and Jha 2017] Singh, S. and Jha, R. K. (2017). A survey on software defined networking: Architecture for next generation network. *Journal of Network and Systems Management*, 25(2):321–374.

[Sood and Xiang 2017] Sood, K. and Xiang, Y. (2017). The controller placement problem or the controller selection problem? *Journal of Communications and Information Networks*, 2(3):1–9.

[Xia et al. 2015] Xia, W., Wen, Y., Foh, C. H., Niyato, D., and Xie, H. (2015). A survey on software-defined networking. *IEEE Communications Surveys Tutorials*, 17(1):27–51.

Capítulo 4

Análise do tráfego de máquinas virtuais na rede de controle de nuvens computacionais baseadas em OpenStack

Charles Christian Miers (UDESC), Guilherme Piêgas Koslovski(UDESC), Maurício Aronne Pillon(UDESC), Adnei Willian Donatti(UDESC)

Resumo

A computação em nuvem é um paradigma amplamente difundido para disponibilização de recursos computacionais sob demanda. Como suporte para o gerenciamento e provisionamento dos serviços ofertados, emprega-se a virtualização dos recursos de processamento, armazenamento e comunicação. Assim, o usuário pode ter acesso a uma infraestrutura virtualizada e privada mediante pagamento pelo uso, ou com a implantação de uma nuvem privada, cuja utilização alinha-se às necessidades de uma organização. A literatura especializada aponta que questões relacionadas ao desempenho de uma nuvem dependem da identificação de comportamentos e operações que ocorrem durante o seu uso. Além disso, percebe-se, que há uma necessidade de identificar operações, muitas das quais ocorrem no nível da camada de rede e demandam análise e compreensão do que está trafegando e sua finalidade. Neste sentido, a caracterização de tráfego auxilia este entendimento, por meio do emprego de técnicas e métodos que possibilitam a coleta e identificação de forma sistematizada. O presente minicurso apresenta a análise do tráfego de máquinas virtuais na rede de controle de nuvens computacionais gerenciadas por OpenStack. Além da descrição e análise do cenário, o minicurso apresenta resultados experimentais obtidos em uma nuvem computacional privada.

4.1. Conceitos básicos

A computação em nuvem (Subseção 4.1.1) possibilita uma forma otimizada e sob demanda de fornecer e consumir recursos computacionais como processamento, armazenamento e rede. Neste contexto, as soluções de nuvem empregam mecanismos de orquestração e gerenciamento para que diversas tecnologias já existentes possam ser empregadas harmoniosamente. Assim, soluções de nuvem empregam sistemas operacionais, técnicas de virtualização e sistemas de arquivos que já existem há décadas mas organizados

e gerenciados de uma forma diferente. A virtualização é um dos principais conceitos, não apenas através das máquinas virtuais (MVs) (Subseção 4.1.2) mas também através de recursos de rede Subseção 4.1.3).

4.1.1. Computação em Nuvem

Existem várias definições sobre computação em nuvem, contudo, devido a ampla adoção, a que mais se aproxima de uma padronização é a definição proposta pelo *National Institute of Standards and Technology* (NIST) [Mell and Grance 2011], que a trata como um modelo de computação que possibilita acesso a um conjunto de recursos computacionais (*e.g.*, armazenamento, rede e serviços) de maneira conveniente, ubíqua e escalável. Ainda, de acordo com o NIST, a oferta do serviço de computação em nuvem pode ser classificada em três categorias: *Infraestructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS). O IaaS permite que o consumidor tenha acesso a recursos de mais baixo nível, como rede, armazenamento e processamento. Neste sentido, o consumidor pode até mesmo controlar o software na MV. Já no PaaS, o consumidor tem acesso a uma plataforma pré-configurada com um conjunto de linguagens e bibliotecas. Neste caso, o consumidor pode controlar as aplicações que serão instaladas por ele, bem como algumas configurações sobre o ambiente. Em relação ao modelo SaaS, as permissões concedidas ao consumidor são de mais alto nível. Então, há acesso às aplicações executando sobre a infraestrutura da nuvem, bem como a algumas configurações específicas para aquela aplicação. É importante ressaltar que variações e especificações das categorias foram propostas pelos provedores de nuvem, *e.g.*, *Function as a Service* (FaaS), *Container as a Service* (CaaS), *Load Balance as a Service* (LBaaS), entre outros.

O NIST também classifica as nuvens computacionais de acordo com seu modelo de implantação, sendo os modelos público e privado os mais comuns. Na nuvem pública a infraestrutura é de uso aberto, e é gerenciada por alguma entidade (*e.g.*, empresa, organização governamental, organização acadêmica). Já as nuvens computacionais privadas operam sobre uma infraestrutura própria, que é mantida pela organização que a possui, *i.e.*, toda a manutenção da nuvem, bem como aspectos de segurança e desempenho são de responsabilidade desta organização. Além disso, as nuvens privadas buscam atender aos propósitos da organização, e são acessíveis somente aos indivíduos com permissão, o que garante à organização total controle sobre os dados ali contidos [Jadeja and Modi 2012]. Dentre os softwares de código aberto que permitem a criação de nuvens públicas e privadas, destacam-se o OpenStack [OpenStack 2019b] e o CloudStack [project 2019]. Sendo o OpenStack a solução de código aberto IaaS mais empregada mundialmente.

Os principais atores envolvidos no modelo de computação em nuvem são: consumidor, que é o utilizador dos serviços da nuvem; provedor, que é a parte responsável por disponibilizar o serviço ao consumidor; auditor, que é responsável por avaliar a nuvem de modo geral (*e.g.,* avaliações de segurança e desempenho); corretor, cuja aplicabilidade se dá no controle de uso, desempenho e distribuição de serviços em nuvem; e operador, que atua como um intermediário entre o provedor e consumidor, de modo a possibilitar a conectividade e transporte de serviços entre eles [Bohn et al. 2011]. A Figura 4.1 ilustra o modelo de referência para computação em nuvem de acordo com o NIST. Especificamente, a Figura 4.2 apresenta o relacionamento entre alguns desses principais atores. Os usuários dos serviços solicitam os recursos para um provedor de infraestrutura. Cada

serviço é oferecido e executado sobre um conjunto de servidores virtualizados.

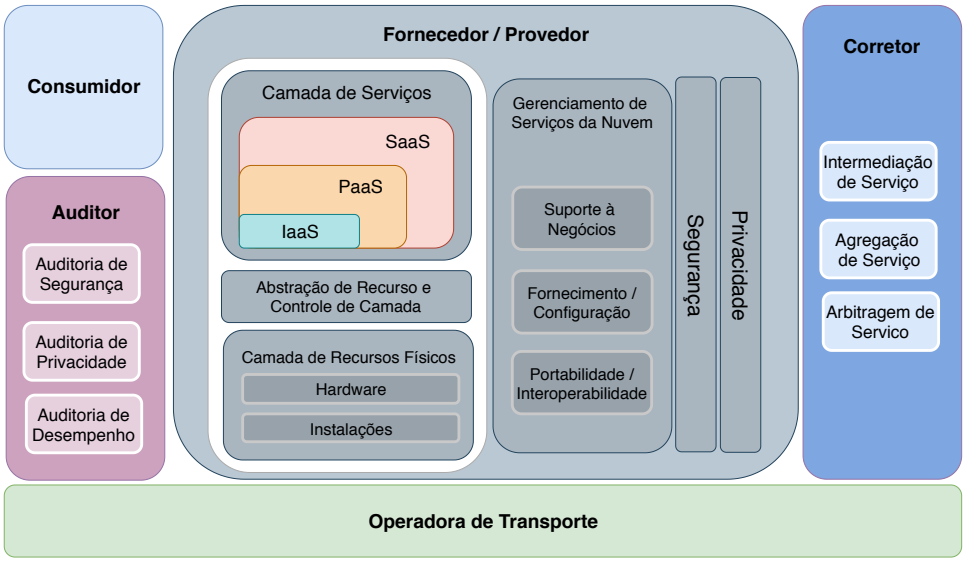


Figura 4.1: Modelo de referência para computação em nuvem de acordo com o NIST [Bohn et al. 2011].

Uma das premissas deste modelo de computação é que vários usuários possam solicitar os seus serviços simultaneamente. Dessa forma, todos os usuários fazem uso da mesma infraestrutura da nuvem (Figura 4.2), que por sua vez, é composta por elementos como servidores, roteadores, *switches* e infraestrutura (*e.g.*, *Ethernet*, fibra ótica e coaxial), que podem causar gargalos que afetam negativamente o fornecimento do serviço.

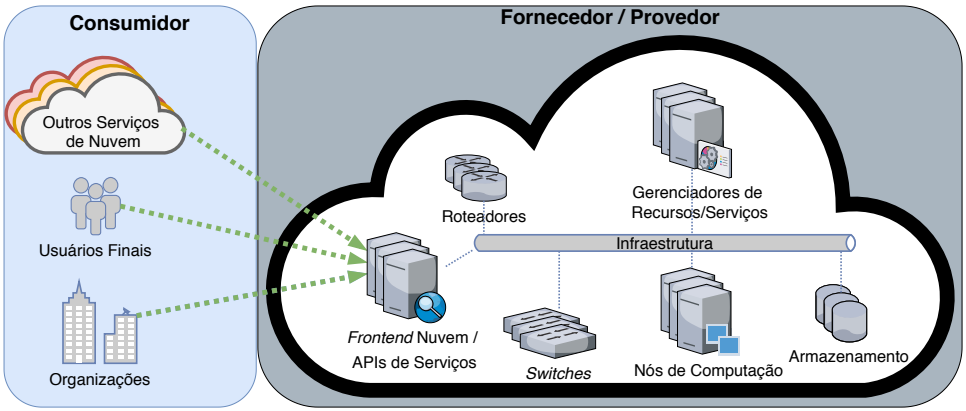


Figura 4.2: Principais atores do modelo de computação em nuvem.

Portanto, independente do modelo, cenário e solução de gerenciamento utilizado, o monitoramento e conhecimento sobre as particularidades da carga computacional e de comunicação são essenciais para garantir qualidade na oferta de serviço e no gerenciamento da nuvem (*e.g.,* configurações de rede e alocação de recursos).

4.1.2. Virtualização de Recursos Computacionais

De acordo com o NIST, a virtualização é a simulação do *software* e/ou *hardware* no qual outro *software* é executado. Além disso, existem diversas formas de virtualização, inclusive, não apenas máquinas virtuais (MVs) podem ser criadas, como contêineres, *virtual appliances* e também na parte de infraestrutura com redes virtualizadas (roteadores, *switches* e VLANs). Neste sentido, um dos principais benefícios para a crescente adoção da virtualização é a eficiência operacional: as organizações conseguem aumentar a carga de trabalho por hardware, uma vez que um único computador pode hospedar diversas máquinas virtuais simultaneamente [Scarfone et al. 2011]. Existem diferentes formas de fornecer recursos virtualizados. Porém, quase toda virtualização tem por objetivo propiciar alguma abstração para um serviço ou uma aplicação. Existem diferentes abordagens para fornecer os recursos computacionais para uma aplicação (Figura 4.3).

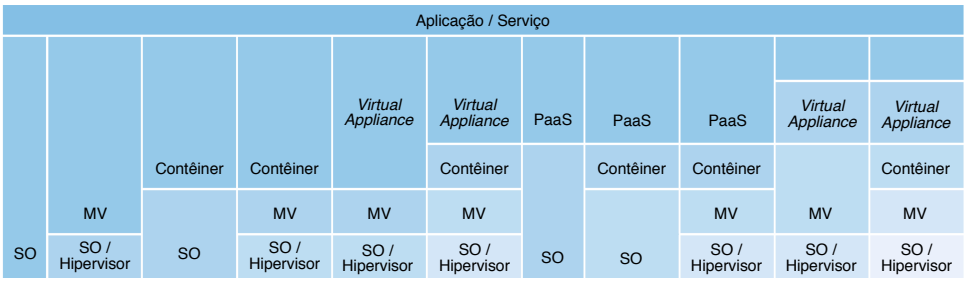


Figura 4.3: Principais formas de prover recursos computacionais a uma aplicação [Panizzon et al. 2019].

Observando a Figura 4.3, constata-se que pode-se usar desde uma abordagem mais antiga com apenas o SO – Aplicação/Serviço até abordagens com SO/Hipervisor – MV – Contêiner – *Virtual Appliance* – Aplicação/Serviço. Apesar de existirem estas diversas combinações, e a adoção de contêineres estar crescendo consideravelmente, o uso de MVs continua muito popular quer na configuração mais tradicional SO/Hipervisor – MV – Aplicação/Serviço ou como base para Contêineres, PaaS ou *Virtual Appliance* [Dawson 2018].

A este ponto, a importância da virtualização já é justificada através da criação de MVs, contudo, sua aplicabilidade na computação em nuvem vai muito além. A virtualização permite criar infraestruturas lógicas, incluindo topologias de redes e computadores, que funcionam sobre os recursos físicos existentes. Dessa forma, além de criar as MVs, também é possível gerenciar o isolamento e a comunicação entre cada uma das máquinas. Geralmente, esta função é feita por algum software, como hipervisores, por exemplo. No caso do OpenStack, que é o foco deste minicurso, os hipervisores comumente utilizados são o QEMU e o KVM. Por fim, embora a virtualização também possa ser realizada com contêineres, o enfoque deste minicurso está na utilização de máquinas virtuais. Um único servidor pode hospedar diversas instâncias (máquinas virtuais), que são gerenciadas por um hipervisor.

Quando o hipervisor é instalado diretamente sobre o *hardware* (sem a existência de um sistema operacional no computador), tem-se a chamada virtualização Tipo 1, ou *bare metal*. Já na virtualização Tipo 2, o hipervisor executa em um sistema operacional.

A Figura 4.4 ilustra a diferença entre os tipos de virtualização. Assim, em ambos os casos o hipervisor é o principal responsável por todas as operações com as instâncias, como criação, que permite escolher um sistema operacional e criar a instância; edição, que possibilita a alteração de algumas configurações de hardware da instância; salvar estado da máquina (*snapshots*), que armazena informações da instância no momento em que foi solicitado, incluindo uso de memória e armazenamento/disco virtual; e exclusão, que permite a remoção total ou parcial da instância [Chandramouli 2014].

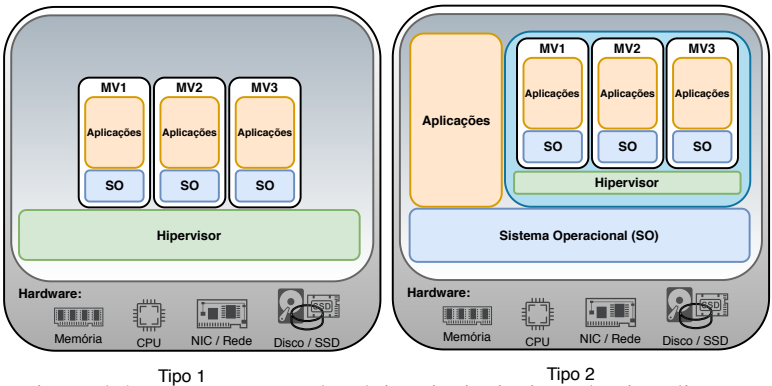


Figura 4.4: Representação dos dois principais tipos de virtualização.

As virtualização possibilitam que diversas MVs sejam hospedadas em um mesmo servidor. Atualmente, hardware para servidores de virtualização possam hospedar dezenas e até mesmo centenas de MVs em um mesmo servidor. Adicionando o fato que as aplicações cada vez tendem a ser mais distribuídas, tem-se a situação na qual a comunicação entre as aplicações/serviços podem ocorrer em MVs que estejam hospedadas em um mesmo servidor e/ou servidores distintos. A Figura 4.5 exemplifica como duas máquinas virtuais podem comunicar-se estando ou não hospedadas no mesmo servidor. Enquanto a comunicação de **MV3** com **MV1** e **MV5** com **MV4** é feita pelo hipervisor, a comunicação de **MV2** com **MV6**, além de passar pelo hipervisor, também passa pela rede física.

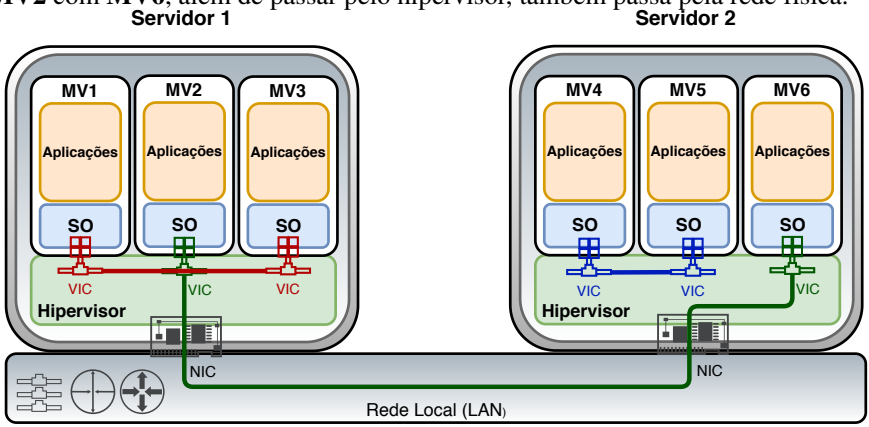


Figura 4.5: Comunicação entre máquinas virtuais estando ou não no mesmo servidor.

Analisando a Figura 4.5, percebe-se que o hipervisor, neste cenário, não fornece apenas a virtualização de computadores (MVs) mas também precisa fornecer soluções

para comunicação entre as MVs, uma vez que nem sempre haverá um recurso de rede físico entre a MV de origem e destino de uma comunicação.

4.1.3. Virtualização de Recursos de Comunicação

A virtualização de recursos de comunicação é amplamente explorada em ambientes de computação em nuvem [Chowdhury and Boutaba 2010]. A Figura 4.5 representa a comunicação entre máquinas virtuais, hospedadas ou não em um mesmo servidor. Quando hospedadas em um mesmo servidor, a comunicação entre as MVs pode ser realizada utilizando roteadores, *switches*, interfaces de rede, *bridges* e endereços *Media Access Control* (MAC) virtualizados, bem como tecnologias tradicionais como *Virtual Local Area Network* (VLAN) e tunelamento *Internet Protocol* (IP). Especificamente, em nuvens OpenStack é recorrente a utilização de *Open vSwitch* [Pfaff et al. 2015] como suporte para a comunicação entre MV.

Em paralelo, o gerenciamento dos recursos de comunicação, virtualizados ou não, podem ser realizados através do uso de *Software Defined Networking* (SDN) [Kreutz et al. 2015, McKeown et al. 2008]. A separação entre os planos de controle e de gerenciamento introduzida por SDN permitiu a composição de infraestruturas virtuais privadas, gerenciadas pelos usuários. Em nuvens OpenStack, o módulo de serviço Neutron faz uso de uma abordagem *Networking as a Service* (NaaS), que através de SDN, fornece serviço de redes aos ambientes computacionais virtualizados. É importante ressaltar que a virtualização de recursos computacionais está presente no oferecimento de serviços aos usuários finais, bem como no gerenciamento da nuvem.

Por outro lado, a virtualização de recursos de comunicação mostra-se fortemente relacionada a implementação do hipervisor e a capacidade deste em usar outras tecnologias/soluções (*e.g.*, *Open vSwitch*, OpenFlow, ...) de modo integrado e facilmente interoperável. Assim, o processo de monitor e analisar tráfego de rede em nuvens computacionais possui uma complexidade ampliada visto que métodos tradicionais (*e.g.*, configurar portas de eco em *switches*, colocar interfaces de rede em modo promíscuo, *etc.*) podem não coletar todo o tráfego.

4.2. Monitoração, análise e caracterização de tráfego

A caracterização de tráfego é uma tarefa usada a fim de entender e resolver problemas que estejam relacionados ao desempenho em redes de computadores [Dainotti et al. 2006]. Neste contexto de nuvens computacionais, a análise e caracterização de tráfego mostra-se como um importante método no levantamento de informações relativas tanto à segurança quanto ao desempenho da nuvem. Através da caracterização de tráfego pode-se, por exemplo, identificar operações em nível de camada de rede bem como o que está trafegando e sua finalidade. Embora esta técnica ainda seja incipiente no contexto da parte de controle em nuvens computacionais, quando se trata de caracterização de tráfego de aplicações tradicionais, como em servidores web, por exemplo, torna-se amplamente empregada [Williamson 2001, Braun and Claffy 1995, Gill et al. 2007].

As nuvens computacionais em geral têm dois locais nos quais a monitoração e análise do tráfego podem ser empregadas. Em primeiro lugar, a rede dos usuários pode ser monitorada a fim de aplicar controles de privacidade e uso da nuvem. Além disso,

também é possível monitorar o tráfego administrativo da nuvem. Desse modo, alguns recursos, tanto computacionais quanto de redes e comunicação, podem ser melhor dimensionados de acordo com o tráfego administrativo analisado, *e.g.*, aumentar a largura de banda quando necessário.

De maneira geral, o estudo do tráfego é separado em duas etapas: medição e análise [Dainotti et al. 2006]. Na primeira, tem-se a coleta de dados que trafegam na rede, enquanto na segunda, é feita uma análise do tráfego para identificar características relevantes ao problema. No contexto deste minicurso, estaremos coletando e analisando o tráfego gerado na rede administrativa do OpenStack (Seção 4.3) durante o ciclo de vida induzido das máquinas virtuais. O objetivo aqui é identificar, a partir do tráfego gerado, quais serviços do OpenStack estão em execução no momento, bem como, qual o volume de tráfego gerado durante o processo. Assim, estas informações podem ser utilizadas no gerenciamento proativo da rede, como por exemplo, criar uma rede isolada para tráfego das imagens das instâncias.

Durante a etapa de medição de tráfego, podem ser utilizadas diversas ferramentas para capturar os dados que trafegam pela rede (*e.g.*, TCPdump e SNORT), contudo, durante os experimentos aqui descritos, utilizou-se uma ferramenta autoral baseada em TCPdump, que coleta e armazena os pacotes. Além disso, dependendo de como é implementada, a medição pode ser feita de forma passiva ou ativa [Vilela 2006, Williamson 2001]. O método passivo é aplicado em casos onde não se deseja intrusão na rede. Não são feitas injeções de tráfego ou quaisquer outras alterações na rede. Já no método ativo, o tráfego na rede pode ser influenciado pela ferramenta de medição.

Em relação a etapa de análise, aplicam-se técnicas de análise de tráfego, como a modelagem estatística simples, que permite analisar atributos genéricos do tráfego e correlacioná-los com as aplicações correspondentes que estão em execução [Nguyen and Armitage 2008]. Neste sentido, são utilizados atributos como as portas de origem e destino dos pacotes, que indicam o serviço correspondente a estes pacotes. Além disso, é importante notar que o processo de caracterização de tráfego é consideravelmente variável de acordo com o ambiente (configuração de rede). Dessa forma, as Seções 4.3 e 4.4 trazem informações sobre o funcionamento do OpenStack, bem como, o ambiente no qual foram realizados os experimentos.

4.3. OpenStack

O OpenStack é um conjunto de *software* e ferramentas de código aberto para criar e gerenciar nuvens computacionais públicas e privadas do tipo IaaS. De acordo com o OpenStack Project [OpenStack 2019b], trata-se de um sistema operacional para nuvens, que controla um amplo grupo de recursos de computação, armazenamento e rede por todo o *data center*. Atualmente, o OpenStack está na sua 19ª versão (denominada Stein).

O OpenStack tornou-se uma solução de nuvem amplamente utilizada no mundo devido a quantidade de empresas envolvidas no seu desenvolvimento (`https://www.openstack.org/analytics`). Tal fato fez com que as principais soluções de MVs (*e.g.*, KVM, QEMU, VMware ESX/ESXi, Citrix Xen, Microsoft Hyper-V, UML, Virtuozzo, IBM zVM, ...) fossem incorporadas aos seus serviços de computação. Da mesma forma foram incorporados serviços de virtualização de comunicação (*e.g.*, SDN com

OpenFlow, ...) e armazenamento (*e.g.*, Gluster, CEPH, ...).

4.3.1. Serviços OpenStack

Os serviços do OpenStack, usados para o provisionamento de nuvens computacionais, são separados por módulos. Além disso, é possível que módulos opcionais sejam acoplados a uma instalação. A Figura 4.6 mostra todos os principais módulo de serviço do OpenStack, bem como serviços e recursos externos que podem ser incorporados a solução de nuvem

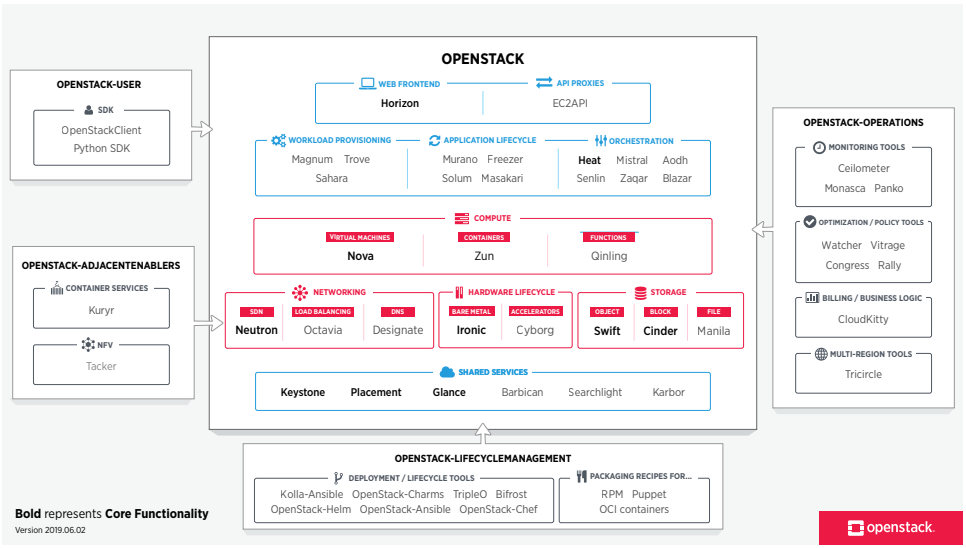


Figura 4.6: Visão geral dos módulo de serviço disponíveis no OpenStack [OpenStack 2019a]

Módulos mais comuns do OpenStack presentes nas instalações tradicionais:

- **Horizom**: Disponibiliza um serviço de *dashboard web* para uso e administração da nuvem;
- **Keystone**: Disponibiliza um serviço de autenticação e autorização para outros serviços do OpenStack;
- **Nova**: Responsável pela distribuição e gerenciamento das instâncias. Realiza tarefas como iniciação, escalonamento e desalocação de MVs;
- **Neutron**: Fornece conectividade de rede entre os outros serviços, bem como disponibiliza uma *Application Programming Interface* (API) para que os consumidores configurem suas redes;
- **Glance**: Atua no processo de armazenamento e recuperação das imagens utilizadas nas MVs;
- **Swift**: Responsável pelo armazenamento e recuperação de objetos não estruturados. Usa técnicas de replicação de dados para tolerância de falhas; e

- **Cinder**: Provê armazenamento persistente em bloco para instâncias em execução.

A Figura 4.7 exemplifica como os principais módulos do OpenStack interagem entre si e com uma MV. Pode-se concluir que, de acordo com a definição da instalação da nuvem, alguns deste módulos podem estar em um mesmo servidor ou em servidores distintos.

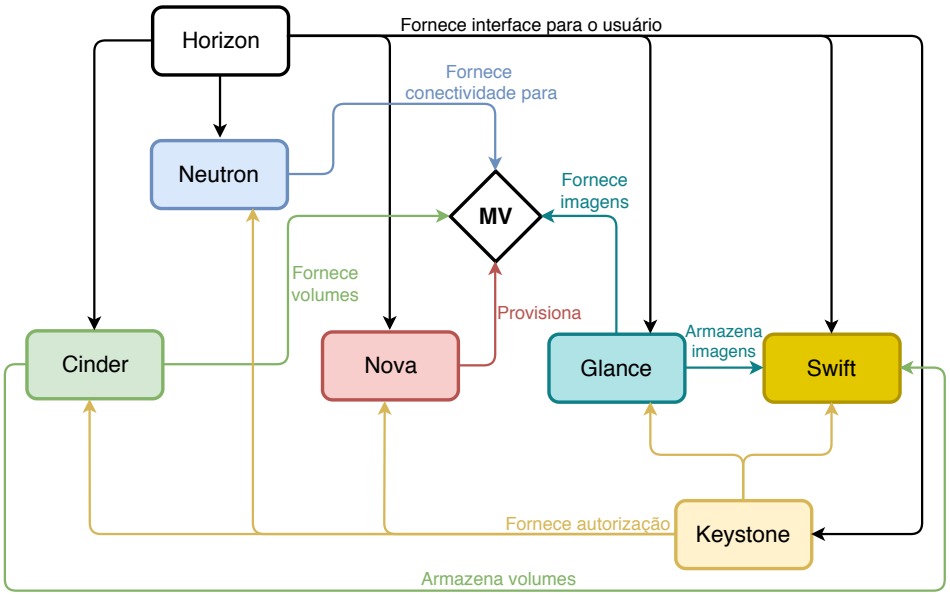


Figura 4.7: Principais operações dos módulo sobre uma MV.

Em implantações do OpenStack em cenário de produção, os módulos são distribuídos entre os servidores da nuvem de modo que existam ganhos com desempenho e segurança da nuvem. Ou seja, serviços computacionalmente intensivos são posicionados em servidores distintos buscando diminuir a interferência no desempenho final do OpenStack. Já em implantações menores, como em uma prova de conceito, os módulos acabam centralizados em um mesmo servidor e os demais servidores são destinados a serviço de computação (*e.g.*, Nova).

4.3.2. Configuração de Rede

Para realizar a comunicação entre os servidores que hospedam diferentes serviços são utilizadas múltiplas redes físicas e virtuais. Neste sentido, existem três domínios criados de acordo com políticas de segurança distintas [OpenStack 2019b]: Domínio Público, Domínio de Controle e Domínio de Convidados. A Figura 4.8 exemplifica a infraestrutura de comunicação recomendada como padrão pelo OpenStack. Podem ser utilizadas técnicas de virtualização de redes, como VLAN, para separação dos domínios de rede.

Domínio Público: são englobadas as redes Externa e de API, que são responsáveis pela visibilidade da API da nuvem criada na Internet e do acesso à Internet pelas MVs. Todos

os endereços IPs das redes devem ser visíveis a partir da Internet para seu pleno funcionamento [OpenStack 2019b].

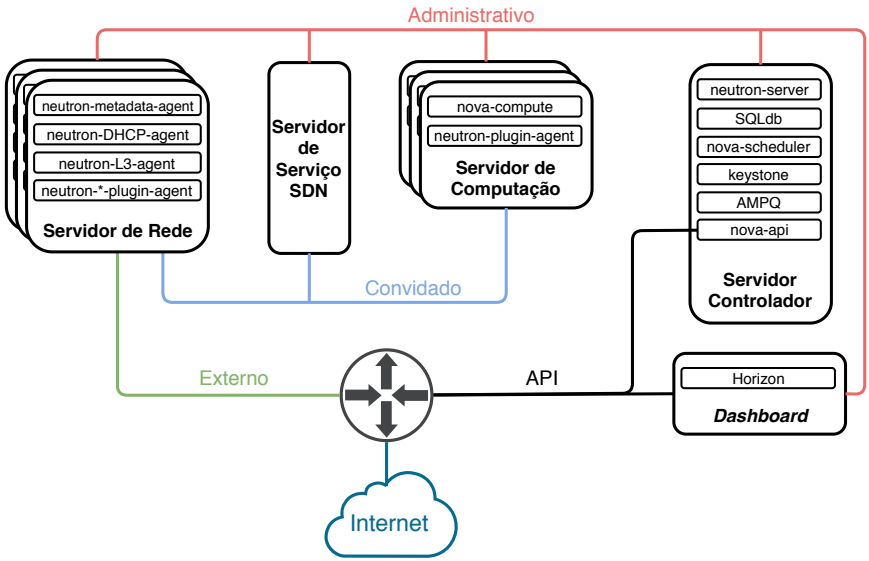


Figura 4.8: Configuração de rede recomendada pelo OpenStack [OpenStack 2019a].

Domínio de Convidados é formado pela rede de comunicação entre as MVs. Os consumidores do serviço de oferta de MVs são referenciados como convidados por não possuírem vínculo direto com a administração da nuvem, logo, não há garantias sobre quem estes convidados são e quais os seus objetivos. Além disso, através do serviço Neutron, são utilizadas tecnologias de virtualização de rede tanto para isolar o tráfego entre as diversas MVs hospedadas em um servidor, quanto para criar redes virtuais entre elas e tornar possível a comunicação quando assim configurado pelo consumidor [OpenStack 2019b].

Domínio de Controle: é formado principalmente pela rede responsável pelo tráfego de controle do OpenStack, que corresponde ao tráfego gerado pela comunicação entre os seus serviços. Esta é a rede mais interna da nuvem, sendo que somente os administradores devem acessá-la. Portanto, esta rede dispensa o uso de técnicas de virtualização da rede. Além disso, tanto os servidores de controle quanto os de computação devem ter uma interface física de rede que conecta-se com a rede de controle. O uso de múltiplas interfaces de rede permite a conectividade à rede de controle e as outras redes conforme o caso, como a rede de convidados no caso dos servidores de computação. Logo, em casos normais, uma MV em execução nunca poderá acessar a rede de controle. [Krutz and Vines 2010].

4.3.3. Fluxo de Dados para Criação de Máquinas Virtuais

O OpenStack permite que sejam feitas diversas operações com as instâncias de MVs, como criar, excluir e armazenar. Toda vez que o usuário realizar uma operação com a instância, o estado dela é alterado de acordo com a operação realizada. Por exemplo, ao remover uma instância, será aplicado o estado DELETED quando a operação for completada. Neste sentido, a documentação fornece uma lista com todas as possíveis operações e estados associadas às instâncias, contudo, neste minicurso serão utilizadas as operações principais para manipulação de uma MV: *CREATE()*, *SUSPEND()*, *RESUME()*, *STOP()*

e *SHELVE()* [OpenStack 2019b].

A Figura 4.9 mostra os possíveis estados que uma instância de máquina virtual pode assumir de acordo com as operações realizadas. Com exceção de BUILDING, que representa a construção inicial da instância, todos os demais estados apresentados nesta figura são chamados de "VM_STATE", pois, indicam o estado estável e atual da instância. Dessa forma, o estado ACTIVE, que indica que a instância está em execução com a sua respectiva imagem, será atingido após as operações *CREATE()* e *RESUME()* serem executadas. Já o estado SUSPENDED, indica a suspensão da instância em nível de hipervisor, ou seja, o estado atual da máquina é mantido, mas é salvo na unidade de armazenamento, não em memória, como é o caso do estado PAUSED. Em relação ao STOPPED, a máquina encontra-se parada e, todos os recursos associados a ela são desalocados. Isso indica que a instância não pode ser restaurada no seu estado anterior. Quanto ao SHELVED, este estado indica que a MV encontra-se desligada e armazenada para uso posterior. Assim, todos os dados e recursos relacionados a ela são mantidos, como volumes adicionados, por exemplo, contudo, as informações em memória são descartadas [OpenStack 2019a].

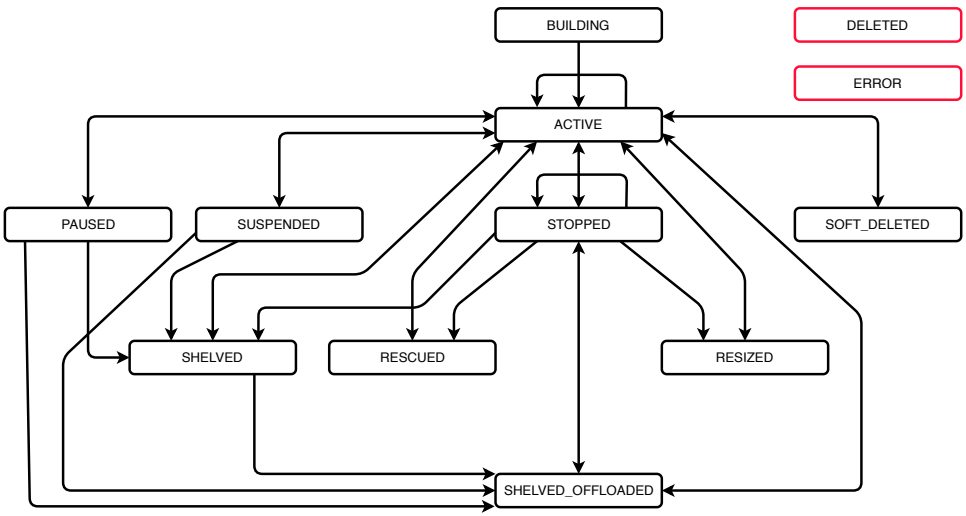


Figura 4.9: Transições de estados para o gerenciamento de uma MV em OpenStack [OpenStack 2019a]

O OpenStack também fornece o estado das tarefas das instâncias, chamado de "TASK_STATE", que não deve ser confundido com o "VM_STATE". O estado da tarefa sempre indica uma ação em execução no momento (transição de estado), enquanto o "VM_STATE"indica uma ação que já foi concluída (estado estável atual da MV). Por exemplo: a operação *SUSPEND()* aplica um "TASK_STATE" SUSPENDING e, ao finalizar, um "VM_STATE"SUSPENDED. Além disso, quando não há nenhuma tarefa sendo executada na instância, aplica-se o "TASK_STATE"*None* a ela. Durante o minicurso será dado maior enfoque ao "VM_STATE", ainda assim, é importante entender as diferenças entre o estado da tarefa e da MV.

Em termos de comunicação e volume de tráfego gerado na rede, as transições

de estado mais custosas são aquelas que requisitam ou persistem dados relacionados às instâncias de MVs. Dessa forma, de acordo com a configuração mostrada na Figura 4.8, o servidor de computação, no qual a instância será hospedada, encontra-se separado dos demais, o que significa que a comunicação pela rede é necessária. Neste sentido, como neste caso não existe uma rede dedicada para armazenamento (*Storage*), além de toda comunicação inter-serviço, a rede administrativa será utilizada como meio de transmissão das imagens das instâncias, por exemplo. Então, operações como *CREATE()* e *SHELVE()* têm maior custo, no sentido de haver maior comunicação entre os serviços e gerar mais tráfego na rede administrativa.

Por fim, para entender melhor o fluxo de dados para criação de máquinas virtuais, alguns processos internos do OpenStack devem ser conhecidos. Inicialmente, é necessário uma imagem do sistema operacional que será executado na instância. Então, conforme a instalação do OpenStack, essa imagem será armazenada no Swift ou no Glance, que serão encarregados por distribuí-la ao servidor de computação quando assim requisitado. Além disso, outro processo importante é o de autenticação e autorização na nuvem, que é realizado pelo Keystone, como descrito na Subseção 4.3.1. Sem autorização prévia, a criação da instância não é permitida. Também é válido notar que, durante o provisionamento da instância, existem trocas de mensagens entre os serviços do OpenStack, como chamadas API feitas para requisitar informações de outros módulos, por exemplo.

4.3.4. Monitoração em OpenStack com Foco na Rede de Controle

A maior parte das pesquisas preocupa-se em analisar as redes da nuvem do ponto de vista dos usuários, deixando de fora toda a parte de operações internas e comportamento do provedor da nuvem [Aishwarya. K and Sankar 2015, Chen and Zhao 2012, Shete and Dongre 2017]. O dimensionamento e controle de tráfego na rede administrativa de uma nuvem OpenStack depende da estratificação de uso e, consequente impacto de operações de controle oriundas de demandas de ações dos usuários. A ausência de monitoração ou caracterização de tráfego na rede administrativa de uma nuvem OpenStack pode causar colapsos e gargalos na rede degradando a qualidade ou mesmo a indisponibilidade do serviço. Entre as operações mais comuns de um usuário de nuvem estão as de gerência de MV, foco de análise de trabalho. Portanto, o tráfego coletado é oriundo das operações criação, remoção, suspensão, etc em MVs. Para isso, seguiu-se um ciclo de vida induzido das instâncias, que as faz transitar entre os estados BUILDING, ACTIVE, SUSPENDED, STOPPED e SHELVED. A Figura 4.10 ilustra este ciclo, bem como mostra as operações realizadas para cada transição de estado. Por fim, será possível dizer quanto tempo cada operação levou para executar, qual o volume de tráfego gerado por ela e quais chamadas de API foram feitas durante o processo.

4.4. Ambiente de testes

Para que o objetivo final de análise e caracterização do tráfego seja atendido, os testes consistem em submeter instâncias a um ciclo de vida induzido, baseado nas operações mais comuns em máquinas virtuais, que consiste em: criação; suspensão; reativação (*RESUME()*); parada; e armazenamento (*SHELVE()*). Dessa forma, cada operação resultará em um volume de tráfego gerado no domínio de controle do OpenStack. Assim, através de uma ferramenta autoral baseada em TCPdump, pode-se coletar e posteriormente

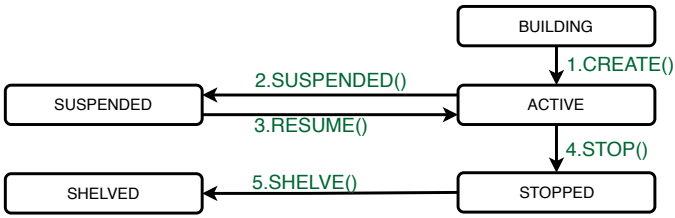


Figura 4.10: Ciclo de vida induzido das máquinas virtuais.

analisar esse tráfego, de modo a identificar o que está trafegando na rede e sua finalidade.

Todos os experimentos são executados em uma instalação do OpenStack Queens, cuja configuração segue o modelo de prova de conceito com dois servidores (Figura 4.11). A infraestrutura da nuvem é provida pelo CloudLab (`https://cloudlab.us/`), que fornece dois servidores, um para computação e outro para controle da nuvem. Cada um dos servidores conta com 256GB de memória e dois processadores que totalizam 16 núcleos [CloudLab 2019].

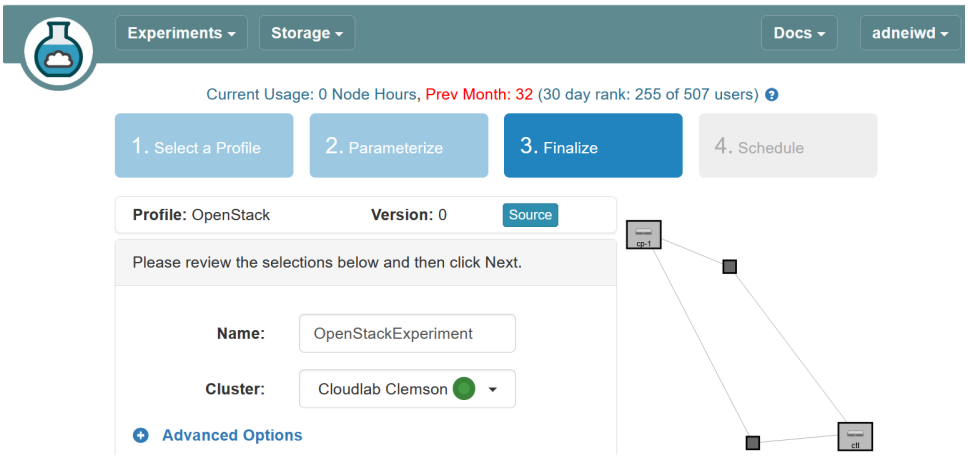


Figura 4.11: Ambiente do CloudLab para criação de um experimento com o OpenStack [CloudLab 2019].

4.5. Estudo de caso com o OpenStack

Inicialmente, os experimentos aqui descritos foram aplicados em uma instalação do OpenStack Queens em um ambiente provido pelo CloudLab. Cada experimento consiste em: (1) identificar a rede administrativa; (2) executar o ciclo de vida induzido das MVs; (3) coletar o tráfego administrativo durante toda a execução; (4) gerar um relatório de transição de estados das máquinas; e (5) analisar o tráfego referente a cada operação.

Na Etapa 1, identifica-se a rede através de testes manuais. Cria-se uma instância na nuvem e verifica-se o volume de tráfego gerado em cada uma das VLANs. A imagem da instância irá trafegar pela rede administrativa. Dessa forma, quando a instância é

criada, a VLAN administrativa terá volume de tráfego próximo ao tamanho da imagem da instância. Como neste ambiente existem apenas duas VLANs, a outra será a pública.

A Figura 4.12 mostra a configuração do ambiente. As Etapas 2 e 3 são feitas por uma ferramenta autoral implementada em Python 3.6, na qual cada operação do ciclo de vida induzido das instâncias é feita através das APIs para Python do OpenStack. Em relação à coleta de tráfego, a ferramenta baseia-se na utilização de TCPdump, e todo tráfego referente a interface escolhida é coletado durante a execução do experimento. A técnica de medição aqui adotada é passiva, visto que não há intrusão na rede (Seção 4.2). Contudo, como o tráfego presente na rede é produto das operações executadas pelo algoritmo, o método de medição também pode ser interpretado como híbrido. A Etapa 4 utiliza o banco de dados do OpenStack, mais especificamente do serviço Nova, para identificar com precisão as transições de estado de cada máquina virtual. Então, fica possível classificar o tráfego coletado referente a uma determinada operação. Por fim, na Etapa 5, o tráfego é analisado de modo a identificar os serviços em execução no momento, como conceituado na Seção 4.2 e detalhado na Subseção 4.3.4.

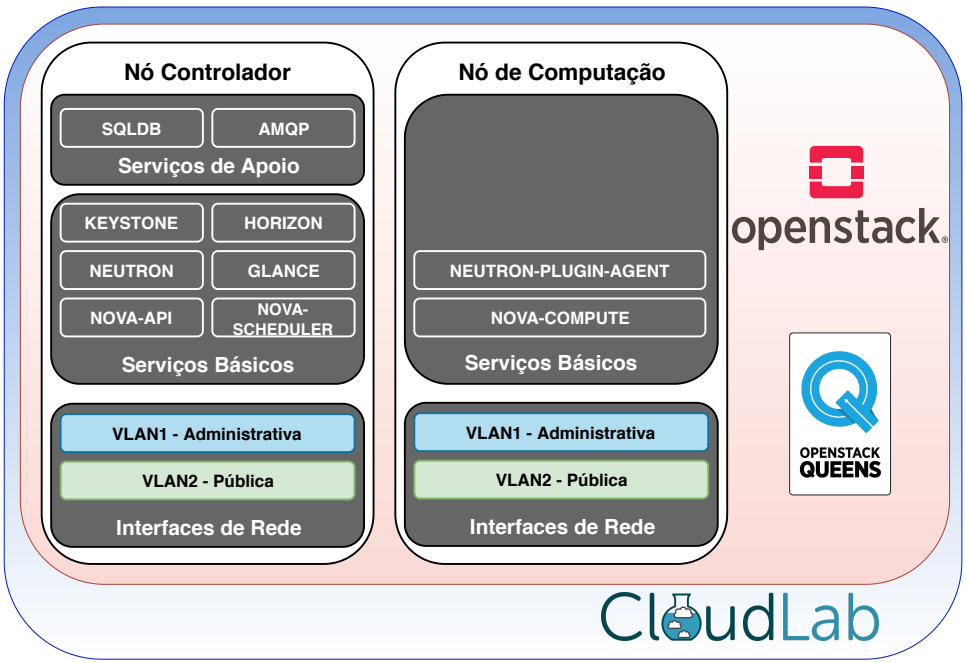


Figura 4.12: Configuração utilizada durante os experimentos, redes Pública e Administrativa são isoladas em VLAN.

4.5.1. Resultados

O processo de experimentação explicado foi aplicado utilizando uma imagem do sistema operacional GNU/Linux CentOS 7, com formato "QCOW2"e tamanho de 1.3GB. O gráfico da Figura 4.13 mostra uma linha do tempo da execução do experimento. A criação da instância de MV é iniciada no segundo 1 e finalizada no segundo 35; a operação

SUSPEND() ocorre entre os segundos 36 e 42; já a operação *RESUME()* se dá entre os segundos 43 e 44; enquanto a *STOP()* vai do segundo 45 até 46; e, a operação *SHELVE()* é executada entre os segundos 47 e 81.

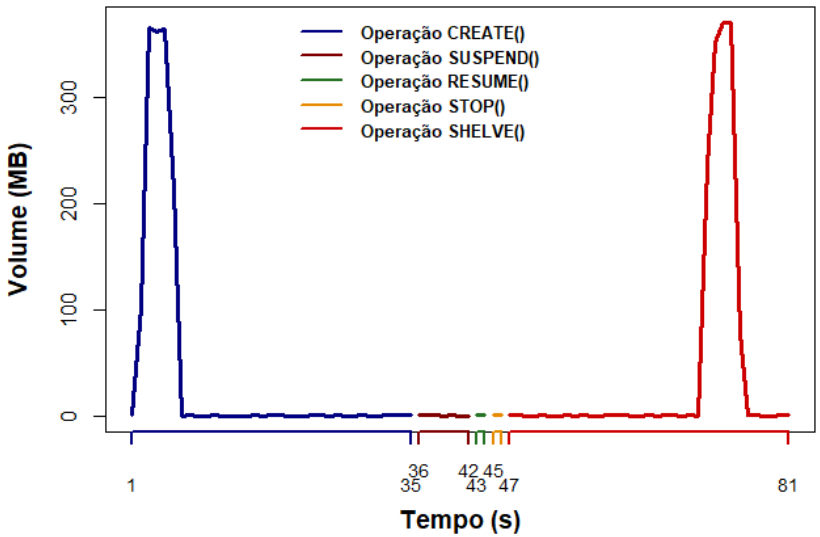


Figura 4.13: Tráfego coletado por segundo durante toda a execução do experimento.

Dessa forma, na Figura 4.13, nota-se que as operações *CREATE()* e *SHELVE()*, além de terem maior período de duração, também são as que produzem maior volume total de tráfego. Além disso, são observados dois picos, um entre 2 e 5 segundos, na operação *CREATE()*, e outro na operação *SHELVE()*, entre 71 e 75 segundos. O somatório de tráfego no intervalo de ambos os picos é de cerca de 1.4GB, o que indica que, muito provavelmente, durante esses intervalos ocorreram as transferências da imagem utilizada na máquina virtual (GNU/Linux CentOS 7) e dos demais recursos associados.

Relacionando os dados do gráfico da Figura 4.13 com a Tabela 4.1, é possível notar que o serviço Glance, que é responsável pelo gerenciamento das imagens das instâncias é o maior responsável pelo volume de tráfego gerado, o que reforça a explicação aos intervalos de pico no gráfico. Na sequência, tem-se a categoria de tráfego "ETC", assim chamada pois, engloba o tráfego de rede referente a serviços diversos, que em sua maioria, é composto pelo RabbitMQ, um software utilizado para troca de mensagens entre os serviços do OpenStack.

Tabela 4.1: Média de tráfego por serviço coletado na interface da rede administrativa do OpenStack. Valores em MB.

	Glance	**Nova**	**Keystone**	**Neutron**	**ETC**	**Total**
CREATE	1398,116	0,003749	0,082098	0,032799	21,04559	1419,28
SUSPEND	0,000208	0,00117	0,011826	0,000104	3,851485	3,864793
RESUME	0	0,001169	0	0,007377	1,453582	1,462128
STOP	0	0,00117	0,011781	0	1,274953	1,287904
SHELVE	1400,127	0,001377	0,023681	0,014858	19,64397	1419,811

Ao analisar o tráfego coletado durante o experimento, também notou-se a presença de diversas chamadas de API do OpenStack. Como mostra a Tabela 4.2, durante a operação de criação da instância, foram realizadas 3 chamadas ao serviço Nova, 13 ao Neutron, 3 ao Keystone e 1 ao Glance. Em relação às operações SUSPEND e STOP, ambas tiveram apenas uma chamada ao serviço Keystone e uma chamada ao Nova. Já a operação RESUME teve apenas uma chamada de API, que foi feita ao Nova. Enquanto a operação SHELVE foi a que teve maior número de chamadas API, com 10 chamadas ao Glance, 3 ao Keystone, 10 ao Neutron e 1 ao Nova.

Tabela 4.2: Número de chamadas API por serviço de acordo com a operação feita na instância.

	Glance	**Nova**	**Keystone**	**Neutron**
CREATE	1	3	3	13
SUSPEND	0	1	1	0
RESUME	0	1	0	0
STOP	0	1	1	0
SHELVE	10	1	3	10

Por fim, de acordo com o módulo de relatório das MVs, todas as operações foram executadas com êxito e, as operações CREATE e SHELVE foram as duas mais demoradas, levando em torno de 34 segundos de execução cada uma. A operação SUSPEND levou em torno de 6 segundos, enquanto a RESUME e STOP levaram apenas 1 segundo cada uma. A Tabela 4.3 mostra os tempos de execução para cada operação.

Tabela 4.3: Tempo de execução de cada operação em segundos.

Tempo de execução em segundos				
CREATE	**SUSPEND**	**RESUME**	**STOP**	**SHELVE**
34	6	1	1	34

4.6. Considerações

As pesquisas mais recentes mostram um considerável interesse em analisar as redes da nuvem que são visíveis aos usuários. Dessa forma, as operações que acontecem no provedor da nuvem acabam não recebendo a devida atenção. Neste sentido, a falta de informações relacionadas ao desempenho do provedor motiva a análise e caracterização do tráfego gerado pelas operações realizadas sobre instâncias de máquinas virtuais (MVs) na nuvem. Inicialmente, de acordo com os resultados mostrados da Subseção 4.5.1, fica visível que as operações de criação e armazenamento das instâncias fazem maior uso da rede. Neste sentido, para que não existam problemas de *Quality of Service* (QoS) ou SLAs, os provedores devem estar preparados para atender a demanda de criação de múltiplas instâncias ao mesmo tempo. Assim, por meio da análise do tráfego gerado por esta operação, é possível planejar a distribuição e configuração dos recursos de rede da nuvem, de modo que o desempenho não seja afetado pela alta demanda de serviço.

Além disso, os resultados também apontam que a maior parte do volume de tráfego gerado é causado pela transferência da imagem do sistema operacional da instância.

Então, caso necessário, a configuração de rede da nuvem pode ser revisada, com o objetivo de diminuir o tráfego de imagens na rede de controle da nuvem. Os resultados também mostram uma significante quantia de tráfego caracterizado como "ETC", pertencente a serviços diversos. Dessa forma, é necessário estudar quais são os serviços diversos em questão e qual o seu impacto na rede administrativa.

Agradecimentos

Os autores agradecem o apoio do Laboratório de Processamento Paralelo Distribuído (LabP2D) no Centro de Ciências Tecnológicas (CCT) da Universidade do Estado de Santa Catarina (UDESC).

Os autores agradecem o apoio da Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (FAPESC).

Referências

[Aishwarya. K and Sankar 2015] Aishwarya. K and Sankar, S. (2015). Traffic analysis using hadoop cloud. In *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–6.

[Bohn et al. 2011] Bohn, R. B., Messina, J., ai Liu, F., Tong, J., and Mao, J. (2011). Nist cloud computing reference architecture. *2011 IEEE World Congress on Services*, pages 594–596.

[Braun and Claffy 1995] Braun, H.-W. and Claffy, K. C. (1995). Web traffic characterization: an assessment of the impact of caching documents from ncsa's web server. *Computer Networks and ISDN Systems*, 28(1):37 – 51. Selected Papers from the Second World-Wide Web Conference.

[Chandramouli 2014] Chandramouli, R. (2014). *Security recommendations for hypervisor deployment*. US Department of Commerce, National Institute of Standards and Technology.

[Chen and Zhao 2012] Chen, D. and Zhao, H. (2012). Data security and privacy protection issues in cloud computing. In *2012 International Conference on Computer Science and Electronics Engineering*, volume 1, pages 647–651.

[Chowdhury and Boutaba 2010] Chowdhury, N. M. K. and Boutaba, R. (2010). A survey of network virtualization. *Computer Networks*, 54(5):862–876.

[CloudLab 2019] CloudLab (2019).

[Dainotti et al. 2006] Dainotti, A., Pescape, A., and Ventre, G. (2006). A packet-level characterization of network traffic. In *2006 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks*, pages 38–45.

[Dawson 2018] Dawson, M. (2018). Red Hat Global Customer Tech Outlook 2019: Automation, cloud, & security lead funding priorities.

[Gill et al. 2007] Gill, P., Arlitt, M., Li, Z., and Mahanti, A. (2007). Youtube traffic characterization: A view from the edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 15–28, New York, NY, USA. ACM.

[Jadeja and Modi 2012] Jadeja, Y. and Modi, K. (2012). Cloud computing - concepts, architecture and challenges. In *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, pages 877–880.

[Kreutz et al. 2015] Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

[Krutz and Vines 2010] Krutz, R. L. and Vines, R. D. (2010). *Cloud security: A comprehensive guide to secure cloud computing*. Wiley Publishing.

[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. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

[Mell and Grance 2011] Mell, P. M. and Grance, T. (2011). Sp 800-145. the nist definition of cloud computing.

[Nguyen and Armitage 2008] Nguyen, T. T. T. and Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*, 10(4):56–76.

[OpenStack 2019a] OpenStack (2019a). Openstack documentation.

[OpenStack 2019b] OpenStack (2019b). What is openstack?

[Panizzon et al. 2019] Panizzon, G., Battisti, J. H. F., Koslovski, G. P., Pillon, M. A., and Miers, C. C. (2019). A Taxonomy of container security on computational clouds: concerns and solutions. *Revista de Informática Teórica e Aplicada*, 26(1):47–59.

[Pfaff et al. 2015] Pfaff, B., Pettit, J., Koponen, T., Jackson, E. J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., and Casado, M. (2015). The design and implementation of open vswitch. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 117–130, Berkeley, CA, USA. USENIX Association.

[project 2019] project, A. C. (2019). Apache cloudstack open source cloud computing.

[Scarfone et al. 2011] Scarfone, K., Souppaya, M., and Hoffman, P. (2011). Guide to security for full virtualization technologies. *NIST Special Publication*, 800:125.

[Shete and Dongre 2017] Shete, S. and Dongre, N. (2017). Analysis amp; auditing of network traffic in cloud environment. In *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 97–100.

[Vilela 2006] Vilela, G. S. (2006). Caracterização de tráfego utilizando classificação de fluxos de comunicação. Mestre em ciências em engenharia de sistemas e computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil.

[Williamson 2001] Williamson, C. (2001). Internet traffic measurement. *IEEE Internet Computing*, 5(6):70–74.

Capítulo 5

Introdução à Web Application Firewalls (WAFs): Teoria e Prática

Felipe Melchior, Diego Kreutz, Maurício Fiorenza, Fernando Flora (Unipampa), Isadora Ferrão (USP), Rafael Fernandes, Thiago Escarrone (Unipampa), Douglas Macedo (UFSC)

Estudos recentes apontam que cerca de 50% das aplicações Web disponíveis na Internet possuem pelo menos uma vulnerabilidade de alta criticidade, como *SQL Injection* [Acunetix 2019]. Se for levado em consideração o nível de risco médio, cerca de 90% das aplicações podem ser consideradas vulneráveis. Ainda segundo relatórios especializados, a vulnerabilidade de *Cross-Site Scripting* (XSS) é uma das mais comuns e mais exploradas (representando cerca de 30%) em aplicações Web [HackerOne 2019]. Além de ser frequente, em alguns casos, a exploração da vulnerabilidade XSS permite ao atacante acessar recursos e dados privados de empresas [Cimpanu 2019].

Como uma forma de contribuir para melhorar o cenário crítico das aplicações Web disponíveis na Internet, especialistas em segurança da informação criaram a fundação OWASP (*The Open Web Application Security Project*)[1]. A entidade tem como principal objetivo disseminar conhecimento sobre segurança de aplicações Web disponíveis na Internet. Além disso, a OWASP também mantém um ranking tri-anual das 10 vulnerabilidades mais recorrentes em sistemas Web.

Segundo a classificação da OWASP de 2017[2], as vulnerabilidades de injeção de falhas (e.g., *SQL Injection*, *NoSQL Injection* e *LDAP Injection*) estão no topo do ranking. Outra vulnerabilidade recorrente em aplicações Web, que aparece na sétima posição, é XSS. A XSS também é um tipo de ataque de injeção de código malicioso nas páginas web. Ao explorar uma vulnerabilidade XSS, o atacante injeta códigos (ou scripts) que executam ações maliciosas como copiar *cookies* e roubar *tokens* de sessão ou quaisquer outros dados do usuário que estão no navegador.

Uma das formas de mitigar o impacto dessas vulnerabilidades antigas e recorrentes, como XSS e *SQL Injection*, é através da utilização de *frameworks* de desenvolvimento

[1] https://www.owasp.org
[2] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

de software como o Laravel[3], que implementa e oferece recursos de segurança que protegem, de forma nativa e transparente, as aplicações Web PHP contra falhas de código. Como exemplo, um *framework* como o Laravel é capaz de, por si só, mitigar a exploração de até 60% das vulnerabilidades mais recorrentes em aplicações PHP [Ferrão et al. 2018].

Outro tipo de ferramenta que tem sido utilizada na proteção de aplicações Web são os *Web Application Firewalls* (WAFs). Um WAF é um serviço de segurança implementado entre o cliente (e.g., navegador/browser) e a aplicação (e.g., sistema PHP rodando num servidor Web Apache). A função do WAF é interceptar e processar as requisições entre o cliente e a aplicação. A partir de um conjunto de regras, ele classifica as requisições em maliciosas (que são geralmente bloqueadas) e não-maliciosas, isto é, que são encaminhadas até a aplicação. Um WAF como o ModSecurity[4], na configuração padrão (isto é, sem nenhuma otimização), associado a um *framework* PHP como o Laravel, é capaz de mitigar em 70% a exploração de vulnerabilidades recorrentes em sistemas Web [Ferrão et al. 2018].

Contra-intuitivamente, a adição de um WAF pode também piorar a segurança de um sistema Web. Pesquisas recentes apresentam casos onde a adição de um WAF ocasiona uma queda (ao invés de um aumento) no nível de proteção da aplicação Web. Este é o caso do cenário utilizando o *framework* PHP Symfony[3] e os WAFs Naxsi[4] e ShadowDaemon[4]. Enquanto o Symfony mitiga em até 60% a exploração das vulnerabilidades mais recorrentes em aplicações Web, a adição dos WAFs Naxsi e ShadowDaemon reduz a mitigação para 50% [Ferrão 2018]. Isto significa que houve uma queda de 10% na segurança da aplicação Web ao adicionar os WAFs.

Os dois principais tipos de WAFs disponíveis no mercado são os que operam como sistemas *standalone* e os SaaS (*Security-as-a-Service*). Os *standalone* são, geralmente, instalados junto aos servidores Web, em que as aplicações estão sendo executadas. Janusec Application Gateway[4], Tempesta FW[4], ModSecurity, OpenWAF[4] e Naxsi são exemplos WAFs *standalone* gratuitos, enquanto que o Imperva[4] e o Citrix[4] são comerciais. Os WAFs SaaS são oferecidos como serviços de segurança sob demanda por terceiros, sendo essencialmente online, oferecidos por empresas como a CloudFlare[5] e a X Labs Security[5]. Neste modelo, o WAF atua como um *proxy* de redirecionamento, onde a requisição é analisada e processada antes de ser encaminhada para o servidor de aplicação propriamente dito.

Apesar de ser um tema relativamente antigo, a importância dos WAFs tem crescido rapidamente no contexto atual, onde ciber ataques, que exploram as vulnerabilidades mais recorrentes de aplicações Web, vêm causando uma quantidade crescente de graves incidentes de segurança [Machado et al. 2019]. Quase que diariamente, incidentes como o comprometimento de sistemas online e vazamento de dados são reportados mundo a fora, levando a prejuízos socioeconômico cada vez maiores [Romani 2016, Convergência Digital 2016, Portinari 2018, Machado et al. 2019]. Em alguns casos, empresas tem

[3] https://laravel.com, https://symfony.com, https://codeigniter.com/, https://phalcon.io/.

[4] https://modsecurity.org, https://github.com/nbs-system/naxsi, https://shadowd.zecure.org, https://www.imperva.com, https://www.janusec.com/, https://github.com/titansec/OpenWAF, https://github.com/tempesta-tech/tempesta, https://www.citrix.com.br.

[5] https://www.cloudflare.com/waf/, https://www.xlabs.com.br/waf/.

entrado com pedido de proteção contra falência devido a incidentes de segurança que levaram a vazamentos de dados, como foi o caso recente da empresa AMCA, nos EUA.

Estudos vêm identificando diferentes desafios e oportunidades de pesquisa no contexto de *Web Application Firewalls*, como algoritmos para detectar ataques que objetivam explorar vulnerabilidades específicas (e.g., *Cross-Site Request Forgery* (CSRF) e XSS) [Srokosz et al. 2018, Rao et al. 2016], mecanismos para aumentar o desempenho de processamento de requisições em cenários com grandes volumes de requisições (e.g., milhares de requisições por segundo) [Moosa and Alsaffar 2008] e revisão minuciosa do estado da arte, procurando comparar e entender o funcionamento de diferentes WAFs [Clincy and Shahriar 2018, Razzaq et al. 2013, Melchior et al. 2019].

O principal objetivo deste minicurso é apresentar um panorama geral da teoria e da prática de *Web Application Firewalls* na proteção de sistemas Web. Alguns dos conceitos básicos e um resumo do estado da arte são apresentados na Seção 5.1. Na sequência, Seção 5.2, são discutidos detalhes técnicos de instalação e configuração de quatro WAFs gratuitos, o ModSecurity, o Naxsi, o ShadowDaemon, o xWAF e o OpenWAF. Nesta seção, são exemplificadas configurações do ModSecurity e do xWAF, bem como formas de verificar na prática o funcionamento de um WAF. Como forma de exemplificar, através de números reais, o impacto de um WAF na latência das requisições Web, na Seção 5.3 são apresentadas e discutidas questões de desempenho dos quatro WAFs gratuitos selecionados anteriormente. Como poderá ser observado, há um impacto significativo na latência das requisições Web, que aumenta de acordo com o número de regras ativas. Além disso, existem diferenças de desempenho, igualmente significativas, entre os WAFs. Finalmente, nas Seções 5.4 e 5.5, são apresentadas uma análise da eficácia de WAFs e *frameworks* de desenvolvimento de software na proteção de aplicações Web e as considerações finais, respectivamente.

5.1. Conceitos Básicos e Estado da Arte

5.1.1. WAFs *standalone* e SaaS

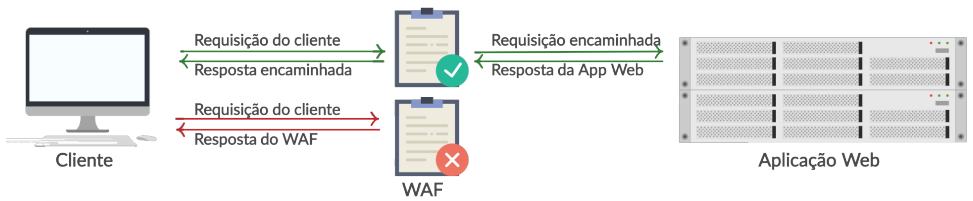
Existem WAFs *standalone*, que são instalados entre o cliente e a aplicação, e WAFs SaaS, que utilizam redirecionamento DNS[6] para que o tráfego seja encaminhado e processado na infraestrutura da empresa proprietária do WAF antes de ser encaminhado ao servidor do sistema Web.

A Figura 5.1 ilustra a utilização e o funcionamento de um WAF *standalone*. Geralmente, este tipo de WAF é instalado no mesmo servidor (i.e., mesma máquina virtual ou física) da aplicação Web que ele irá proteger, reduzindo a latência de comunicação (entre o WAF e o serviço Web) e aumentando a personalização das regras que irão proteger a aplicação. Como será discutido na sequência, criar regras personalizadas, específicas às aplicações Web que serão protegidas pelo WAF, é um dos processo importantes na instalação e manutenção dessas ferramentas de segurança.

Como ilustrado na figura, o cliente realiza uma requisição ao serviço Web (e.g., `links https://unihacker.club`). Esta requisição é recebida e processada pelo WAF, que decidirá se ela segue adiante, até a aplicação Web (e.g., servidor Web Apache2

[6] https://www.pcmag.com/encyclopedia/term/41620/dns

Figura 5.1: Visão geral de um WAF *standalone*



e aplicação PHP), ou se é rejeitada. Caso nenhuma das regras do WAF detecte algo de anormal na requisição, a aplicação Web processa a requisição e envia uma resposta ao cliente. Este processo é ilustrado pelas setas verde da figura. Entretanto, caso alguma das regras do WAF detecte algo suspeito na requisição (e.g., *SQL Injection*), ela será imediatamente negada, isto é, o cliente irá receber uma mensagem de erro do WAF, como ilustrado pelas setas vermelhas na imagem.

O principal desafio no processo de adoção dos WAFs *standalone* é a disponibilidade de recursos humanos qualificados, que são necessários para lidar com os seguintes desafios técnicos:

(dt_1) instalação, configuração e manutenção contínua;

(dt_2) criação e gerenciamento de regras;

(dt_3) otimização de regras; e

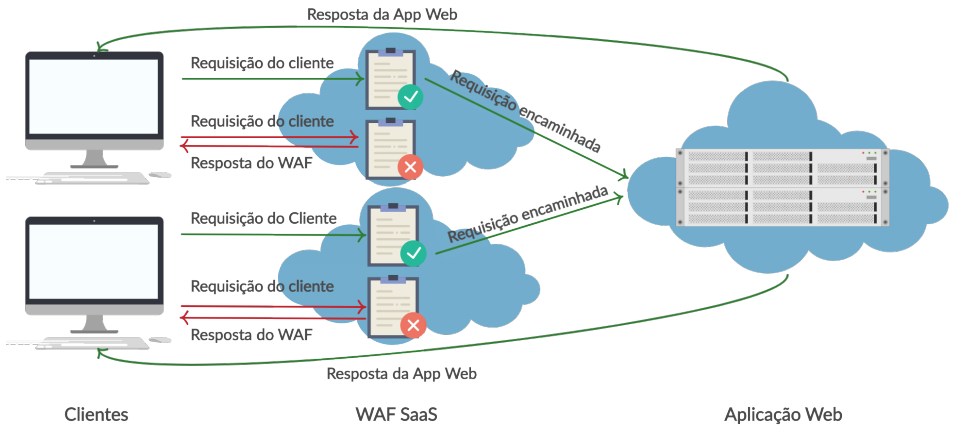
(dt_4) suporte a grandes volumes de tráfego.

Os WAFs SaaS surgem como uma forma de resolver o problema da falta (ou custo) de mão-de-obra especializada, necessária no caso dos WAFs *standalone*. Um WAF SaaS é um serviço oferecido sob demanda e por terceiros, geralmente empresas especializadas em segurança de sistemas. Entretanto, utilizar um WAF SaaS implica em firmar um contrato de confiança entre o contratante e a contratada, pois o tráfego da empresa contratante irá passar pelos sistemas da contratada. Em outras palavras, a confidencialidade e a privacidade dos dados entra em discussão uma vez que fica fácil para qualquer pessoa com acesso à infraestrutura da contratada realizar interceptação de tráfego.

A Figura 5.2 ilustra um WAF SaaS. Enquanto que o controle das requisições é essencialmente idêntico a um WAF *standalone* (i.e., deixa passar ou bloqueia a requisição), há pelo menos dois diferenciais, importantes, que merecem atenção. Primeiro, o tráfego do cliente é redirecionado para a infraestrutura do WAF. Na prática, tomando como exemplo o domínio `unihacker.club` ou `www.xlabs.com.br`, ao invés de o tráfego ir diretamente para a infraestrutura que mantém o domínio, a aplicação Web, o tráfego passa

primeiramente pelo domínio (e.g., `web.armor.zone` da XLabs Security) da empresa que fornece o WAF SaaS. Neste caso, o redirecionamento é realizado diretamente no serviço de DNS do domínio que está contratando o serviço de WAF (e.g., apontar o domínio `unihacker.club` para `web.armor.zone`).

Figura 5.2: Ilustração de um WAF SaaS



O serviço de WAF SaaS é escalável e capaz de atender picos de tráfego, bem como um grande ou variável número de clientes. Para tanto, um WAF SaaS é geralmente construído sobre uma rede CDN (*Content Delivery Network*) ou uma plataforma de computação em nuvem com múltiplos *datacenters* espalhados pelas regiões de atuação ou interesse da empresa do WAF. Como ilustrado na Figura 5.2, há dois clientes utilizando instâncias distintas do WAF. Estas instâncias podem estar na mesma localização geográfica (e.g., mesmo país) ou em localizações geográficas completamente distintas.

5.1.2. Visão Geral do Estado da Arte

A Tabela 5.1 apresenta de forma resumida as principais **contribuições**, **oportunidades** de pesquisa e **evidências** empíricas encontradas em trabalhos relacionados a *Web Application Firewalls*.

Contribuição. Há pesquisas e desenvolvimentos que propõe novos algoritmos para de detecção de ataques a vulnerabilidades específicas (e.g., CSRF e XSS). Por exemplo, ataques que visam explorar vulnerabilidades XSS podem ser bloqueados através da análise e identificação de padrões (e.g. caracteres “<” e “:”) frequentemente utilizados em requisições maliciosas aos sistemas Web [Rao et al. 2016].

Outro assunto importante são as boas práticas na criação de novas regras para o WAF. O primeiro passo é entender o objetivo da regra que está sendo criada [Clincy

Tabela 5.1: Resumo do estado da arte

	Contribuição	Oportunidades	Evidências
[Funk et al. 2018]	WAF com objetivo de melhorar a taxa de detecção de ataques	Comparar com outros WAFs, além do ModSecurity	Taxa de detecção cerca de 60% maior em relação ao detectado pelo ModSecurity
[Srokosz et al. 2018]	Detecção de ataques *CSRF*	Implementar e avaliar os algoritmos	
[Rao et al. 2016]	Detecção de ataques *XSS*	Implementar e avaliar os algoritmos	
[Moosa and Alsaffar 2008]	WAF híbrido para aplicações governamentais	Evoluir o WAF com novas heurísticas	WAF lida bem com um volume grande de requisições
[Razzaq et al. 2013]	Comparação analítica de quinze WAFs tradicionais	Comparar de forma empírica os WAFs e adicionar soluções *SaaS*	
[Clincy and Shahriar 2018]	Boas práticas para criação de regras	Avaliar configurações padrão de WAFs	
[Rietz et al. 2016]	Visão de WAF no estado atual da Internet	Avaliar WAFs em cenários controlados	
[Singh et al. 2018]	Análise dos níveis de Paranoia do ModSecurity	Avaliar diferentes configurações	Níveis Paranoia aumentam a taxa de detecção e de falsos positivos
[Ferrão 2018]	Avaliação de três WAFs em cenários controlados	Analisar outros WAFs e melhorar a taxa de detecção de ataques	ModSecurity mitiga 70% das vulnerabilidades do Top Ten da OWASP

and Shahriar 2018]. Isto é crucial para o bom funcionamento e desempenho de um WAF. Por exemplo, uma regra estática, que bloqueia ataques de *SQL Injection* que utilizam a expressão `"' OR 1=1- #"`, não irá bloquear um ataque que utilize a expressão `"' OR 2=2- #"`.

Oportunidades. A maioria dos estudos indica de forma explicita ou implícita oportunidades de pesquisa e desenvolvimento, como implementar novas heurísticas para melhorar a taxa de detecção de WAFs, avaliar e comparar a qualidade dos WAFs através de estudos empíricos, avaliar soluções SaaS e avaliar empiricamente as di-

ferentes configurações dos WAFs. Neste minicurso, mais precisamente nas Seções 5.3 e 5.4, são apresentados e discutidos dados sobre o impacto de WAFs na latência de requisições Web e na segurança de aplicações Web.

Evidências empíricas são dados concretos que permitem melhor avaliar e validar uma pesquisa. Por exemplo, o WAF ModSecurity, na configuração padrão, utilizando um cenário controlado, é capaz de mitigar até 70% dos ataques que objetivam mitigar as vulnerabilidades mais recorrentes em aplicações Web, conforme dados apresentados na Seção 5.4. Indo um pouco além, resultados de pesquisa indicam que implementações específicas de WAFs podem atingir uma eficácia de detecção de cerca de 60% maior que o ModSecurity [Funk et al. 2018].

Analisando o estado da arte, é possível identificar alguns dos principais desafios encontrados na utilização de *Web Application Firewalls*, como detectar ataques específicos (e.g., CSRF), compreender os aspectos de desempenho de um WAF em diferentes cenários, dominar a ferramenta a ponto de criar regras eficazes e otimizar a utilização dos recursos disponíveis.

Detecção de Ataques. Alguns ataques, como os que exploram CSRF e *Server Side Request Forgery* (SSRF), são difíceis de serem mitigados [Srokosz et al. 2018]. *Tokens* podem ser utilizados nas requisições como uma forma de mitigar este tipo de ataque. Entretanto, um *token* mal formulado pode ser burlado pelo atacante. Para resolver o problema, o WAF pode utilizar autenticação adicional ao detectar uma anomalia (um desvio de padrão) entre a requisição atual e o histórico de requisições do usuário.

Desempenho. Em alguns cenários, o desempenho de WAFs é crítico devido ao grande número de requisições a serem processadas [Moosa and Alsaffar 2008]. Arquiteturas híbridas, ancoradas em heurísticas e redes neurais, podem ser utilizadas para melhorar o desempenho dos WAFs nesse tipo de cenário.

Domínio da ferramenta. Explorar as diferentes configurações de um WAF é importante para que se possa extrair o máximo de eficiência da ferramenta. Tomando como exemplo o ModSecurity, a solução possui os chamados níveis de Paranoia, que são tipos de regras ativadas apenas quando necessário, como durante uma sequência de ataques, por exemplo [Singh et al. 2018]. No nível de Paranoia baixo, o número de falsos positivos é baixo, porém a taxa de detecção também é menor. Ao ativar o nível de Paranoia alto, todas as regras presentes na configuração são ativadas ao mesmo tempo, detectando um número maior de ataques. Entretanto, a taxa de falsos positivos também pode aumentar.

5.2. Instalação, Configuração e Verificação do Funcionamento de WAFs

Nesta seção são apresentadas informações de instalação, configuração e verificação do funcionamento dos WAFs: ModSecurity, Naxsi, ShadowDaemon, xWAF e OpenWAF. A instalação dos WAFs seguiu a respectiva documentação de cada ferramenta. Ferramentas

como ModSecurity, Naxsi e ShadowDaemon seguem uma linha de instalação bem semelhante, podendo mudar alguns detalhes específicos de configuração. O xWAF possui um processo de instalação diferentes dos demais WAFs visto que trata-se de um código PHP que necessita ser incluído no código da aplicação Web. Já o OpenWAF requer a instalação de pacotes e compilação a partir do código fonte.

O ModSecurity será utilizado para exemplificar a instalação de um WAF na forma tradicional. Seus arquivos estão disponíveis nos repositórios de pacotes da maioria das distribuições Linux. Consequentemente, basta executar o comando de instalação de acordo com o gerenciador de pacotes do sistema em questão. Como foram utilizadas máquinas virtuais rodando Linux Ubuntu Server 16.04, o ModSecurity foi instalado através do APT, utilizando o seguinte comando: `apt install libapache2-modsecurity`.

Após a instalação do pacote, o primeiro passo consiste em habilitar o ModSecurity, o que pode ser realizado através da adição da linha `SecRuleEngine On` no arquivo `/etc/modsecurity/modsecurity.conf`. Na sequência, devem ser adicionadas as regras. Seguindo a documentação da ferramenta, é recomendado o uso do pacote de regras CRS (*Core Rule Set*)[7], que consiste em um conjunto de regras pré-definidas. Estas regras, segundo a documentação, mitigam as dez vulnerabilidades mais recorrentes na Web conforme classificação da OWASP.

O passo seguinte consiste em configurar o arquivo `security2.conf`, disponível no diretório `/etc/apache2/mods-enabled/`, que representa a extensão do ModSecurity no Apache2. No arquivo é necessário indicar o diretório (e.g., `/var/cache/modsecurity`) onde serão armazenados os dados (e.g., dados de endereços IP, dados de sessões) persistentes do ModSecurity e a localização do conjunto de regras (e.g., `/usr/share/modsecurity-crs/`) que serão carregadas e utilizadas pelo ModSecurity. O conteúdo final do arquivo é ilustrado a seguir.

```
<IfModule security2_module>
  # Diretório para armazenar os dados persistentes
  SecDataDir /var/cache/modsecurity
  # Inclusão de todos os arquivos de regras do diretório
  IncludeOptional /usr/share/modsecurity-crs/*.conf
</IfModule>
```

Enquanto que o processo de configuração dos WAFs Naxsi e ShadowShadow é similar ao ModSecurity, a configuração do xWAF é diferente. O xWAF é um código PHP que implementa uma classe de funcionalidades e necessita ser instanciada em cada arquivo PHP da aplicação Web, como ilustrado a seguir.

```
<?php
    require('xwaf.php');
    $xWAF = new xWAF();
    $xWAF->start();
```

[7] https://github.com/SpiderLabs/owasp-modsecurity-crs

```
        ... código da aplicação ...
?>
```

As três linhas adicionadas no início do arquivo PHP da aplicação importam o código do xWAF, disponível no GitHub[8], instanciam um novo objeto da classe `xWAF` e invocam o método `start()` do objeto criado, inicializando o WAF propriamente dito. Entretanto, não é viável realizar manualmente a inserção e manutenção das três linhas de código em todos os arquivos do sistema a ser protegido. Para automatizar o processo, basta configurar o arquivo `php.ini` do PHP, adicionando o parâmetro `auto_prepend_file`, como ilustrado a seguir. Este parâmetro irá fazer com que o xWAF seja automaticamente adicionado a todos os arquivos PHP disponíveis no respectivo servidor Web.

```
...
; Automatically add files before PHP document.
; http://php.net/auto-prepend-file
auto_prepend_file = /var/www/html/waf.php
...
```

Diferentemente do ModSecurity e Naxsi, a solução xWAF não necessita de configuração inicial das regras uma vez que possui um conjunto de regras embutido e ativo em seu código fonte. Caso desejado, outras regras podem ser adicionadas através da edição do código do WAF (e.g., arquivo `waf.php`). Já a ferramenta ShadowDaemon necessita que as regras sejam adicionadas manualmente após a instalação, através de uma interface Web própria.

O OpenWAF foi projetado para ser utilizado com conjunto com o servidor Web Nginx[9] e é divido em dois módulos. O primeiro é baseado em ModSecurity e realiza a verificação e aplicação das regras. O segundo módulo realiza a análise comportamental do trafego com o objetivo de identificar ataques realizados por malwares e violação de cookies, por exemplo.

A instalação do OpenWAF é realizada através de download e compilação de três pacotes pacotes (`pcre-8.42.tar.gz`[10], `openssl-1.1.0h.tar.gz`[11] e `openresty-1.13.6.2.tar.gz`[12]) e do código fonte do WAF[13], conforme detalhado na documentação oficial. A adição e o gerenciamento de regras é realizado através da edição dos arquivos disponíveis no diretório `/opt/OpenWAF/conf`, que é o repositório padrão de regras da ferramenta.

Finalmente, após instalados e configurados, os WAFs precisam ser testados. O teste de funcionamento dos WAFs pode ser realizado através da criação de uma simples regra que bloqueia requisições especificas provenientes do comando `curl`. Utilizando

[8] `https://github.com/Alemalakra/xWAF`
[9] https://www.nginx.com/
[10] https://ftp.pcre.org/pub/pcre/pcre-8.42.tar.gz
[11] https://www.openssl.org/source/openssl-1.1.0h.tar.gz
[12] https://openresty.org/download/openresty-1.13.6.2.tar.gz
[13] `git clone https://github.com/titansec/OpenWAF.git`

como exemplo o ModSecurity, a implementação da regra é apresentada no Algoritmo 1. A regra utiliza o arquivo `curl.txt`, que contém uma lista dos `User-Agents` (e.g., *curl/7.64.1*), um por linha, do comando `curl`. O WAF bloqueia (`deny` no algoritmo) e registra (`log`) todas as requisições cujo cabeçalho contém um `User-Agent` listado no arquivo `curl.txt`. Ao bloquear uma requisição, o WAF registra o motivo do bloqueio (`msg`) e os detalhes da solicitação, como URL da requisição e endereço IP do cliente.

Algorithm 1 Bloqueia requisições do `curl`

```
1: SecRuleEngine On
2: SecRule REQUEST_HEADERS:User-Agent "@pmFromFile curl.txt
   id:12345,deny,log,status:403,msg:'curl tentando enviar
   requests'"
```

Caso o usuário envie uma requisição ao servidor (e.g., `curl 192.168.1.1/index.html`), utilizando uma versão do comando `curl` listada no arquivo `curl.txt`, a solicitação será automaticamente bloqueada. O cliente recebe como resposta o código HTTP 403 (`status`)[14], que significa que a solicitação foi entendida pelo servidor, porém não será atendida.

5.3. Impacto dos WAFs na Latência das Requisições Web

Ao instalar um *Web Application Firewall*, o administrador do sistema deve analisar as necessidades da aplicação e configurar o WAF de acordo, incluindo regras específicas para ataques específicos ou aumentando o número de regras ativas, por exemplo [Rao et al. 2016, Clincy and Shahriar 2018]. Enquanto que, por um lado, um número maior de regras ativas pode levar a uma maior capacidade de detecção, por outro lado, pode levar a um impacto na latência das requisições Web.

Com o objetivo de tentar identificar o impacto do número de regras, em diferentes WAFs, na latência das requisições, foi implementado um programa em Python utilizando a biblioteca `Requests`. O programa simula dois tipos de usuários, um não malicioso e outro malicioso. Enquanto que as requisições do primeiro não são bloqueadas (**Pass**), as do segundo são bloqueadas (**Match**) pelo WAF.

Para os testes de desempenho, foram instaladas quatro máquinas virtuais, uma para cada WAF (ModSecurity, Naxsi, ShadowDaemon, xWAF), com 1 vCPU, 2Gb de RAM e a distribuições Linux Ubuntu Server 16.04. A máquina hospedeira possui processador i5 7300-HQ quad-core de 2.5Ghz, 8Gb de memória RAM, placa de vídeo GTX 1050, disco rígido Western Digital, modelo WD10SPZX de um terabyte, controladora HM170/QM170 Chipset SATA de 6.0Ghz, com 5400 rotações por minuto e cache de 128Mb, executando a distribuição Linux Manjaro versão 18.0.4 e o o VirtualBox na versão 6.0.6. Os WAFs foram virtualizados, enquanto que o programa Python foi executado a partir da máquina hospedeira.

A aplicação Web PHP, que implementa as dez vulnerabilidades presentes no ranking da OWASP de 2013[2], conforme proposto por trabalhos recentes [Ferrão 2018], tam-

[14] https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status

bém foi virtualizada junto aos WAFs. Para executar a aplicação, foi utilizado o PHP versão 7.0.3, o MySQL versão 5.7.25 e o servidor Web Apache (versão 2.4.18), exceto no caso do WAF Naxsi, que é compatível apenas com o servidor Web Nginx (foi utilizada a versão 1.13.1).

A Tabela 5.2 resume os tempos médios de latência de uma requisição (em milissegundos). Os valores correspondem a média de um mil requisições. Vale ressaltar que as regras que bloqueiam as requisições dos agentes maliciosos foram inseridas no início do conjunto de regras de cada WAFs. Portanto, os números podem variar caso as regras sejam inseridas em diferentes posições dentro do conjunto de regras do WAF.

Tabela 5.2: Tempo de acesso de acordo com a quantidade de regras

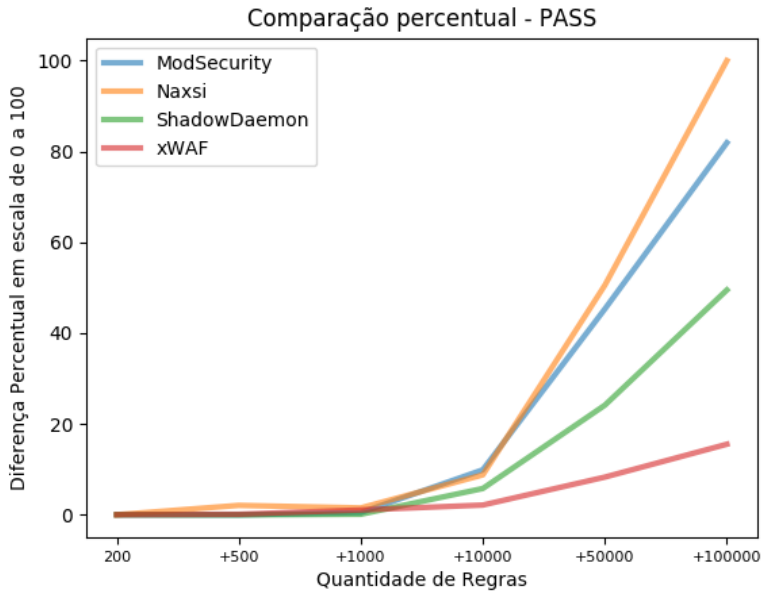
	ModSecurity		**Naxsi**		**ShadowD.**		**xWAF**	
	Pass	**Match**	**Pass**	**Match**	**Pass**	**Match**	**Pass**	**Match**
Padrão	2.70	2.54	1.31	1.01	2.91	2.70	1.31	1.42
+ 500	2.69	2.49	1.90	1.10	2.89	2.74	1.34	1.38
+ 1000	2.98	2.38	1.74	1.23	2.97	2.84	1.61	1.54
+ 10000	8.53	3.39	3.82	3.19	6.58	4.57	1.92	1.77
+ 50000	29.40	6.11	15.76	13.20	18.23	13.04	3.68	2.88
+ 100000	50.97	9.64	29.90	27.39	34.34	26.23	5.75	4.27

Como pode ser observado na Tabela 5.2, existe uma diferença significativa na latência entre os WAFs. Considerando que todos os WAFs foram executados na configuração padrão (linha “padrão” na tabela), a diferença pode ser explicada parcialmente pela quantidade inicial de regras em cada WAF. Por exemplo, o Naxsi inicia com cerca de 70 regras, enquanto que o ModSecurity com aproximadamente 150 regras. Isto, por si só, praticamente já explica a diferença de 2.70ms e 1.31ms (no **Pass**) entre os dois WAFs. Outro motivo provável são os próprios mecanismos de processamento das regras que o WAF implementa.

Os resultados mostram que, por via de regra, o usuário não malicioso acaba sendo o mais prejudicado com o aumento do número de regras ativas. Isto ocorre devido ao fato de uma requisição normal (**Pass**) ser analisada e processada por todas as regras ativas. Por outro lado, uma solicitação maliciosa é bloqueada na primeira regra que identificar o ataque (i.e. primeiro **Match**).

Com o aumento progressivo no número de regras, o ModSecurity passou de 2.70 (170 regras) para cerca de 50ms de latência (100k regras), o que representa um aumento percentual aproximado de 1780%. Entretanto, percentualmente, a solução Naxsi teve um desempenho pior com relação ao aumento do número de regras, variando de 1.31ms (70 regras) para 29.90ms (100k regras), o que representa um aumento percentual de latência na ordem de 2180%, como pode ser melhor observado na 5.3. A Figura 5.4 apresenta o aumento percentual para os casos de **Match**.

Figura 5.3: Aumento percentual do usuário normal (PASS)

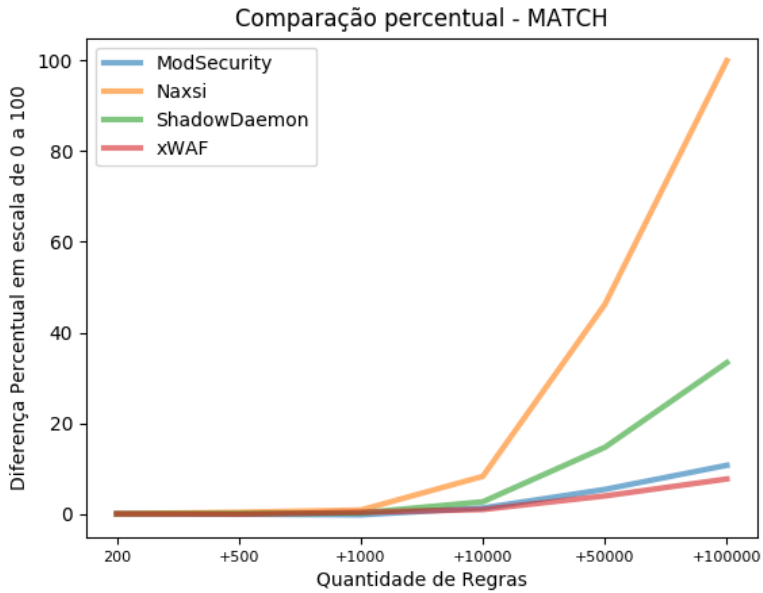


Nos experimentos com requisições bloqueadas pelo WAF (**Match**), a solução Naxsi chegou a uma latência aproximada de 27ms na detecção dos ataques. Adicionalmente, este WAF atingiu também o maior aumento percentual, chegando a 2600%, conforme mostra a Figura 5.4. Enquanto isso, o xWAF aumentou percentualmente a latência das requisições em apenas 200%. Os dados do gráfico nos permitem concluir que os WAFs mantiveram um aumento percentual similar até 10k regras. Entretanto, com mais de 10k regras, a diferença começou a ficar visualmente significativa. Enquanto que o Naxsi e o ShadowDaemon tiveram um aumento exponencial, podemos dizer que os outros dois mantiveram um aumento linear. Estes resultados podem estar relacionados aos mecanismos internos, de processamento da regras, que cada WAF utiliza.

Tanto no caso do usuário não malicioso, quanto no caso do agente malicioso, os menores tempos de latência (6ms) e menores percentuais de aumento (330%), de acordo com o número de regras, foram do xWAF. Esta latência significativamente menor do xWAF, para grandes números de regras, pode ser parcialmente explicado pelo fato de o código do xWAF ser incluído diretamente no código fonte das aplicações PHP. Novamente, temos aqui outro ponto interessante de investigação futura.

Como pode ser observado nos resultados da Tabela 5.2, a partir de cinquenta mil regras adicionais, a latência média das requisições fica próxima de 30ms utilizando o ModSecurity. A título de comparação, considerando uma rede cabeada de fibra óptica, cuja saída do `traceroute` é apresentada a seguir, a latência de acesso ICMP ao servidor

Figura 5.4: Aumento percentual do agente malicioso (MATCH)



de resolução de nomes (DNS) do Google (endereço IP 8.8.8.8) é de cerca de 24ms. Neste exemplo, o WAF está simplesmente dobrando o tempo de acesso a um serviço similar ao DNS da Google.

```
   $ traceroute to 8.8.8.8 (8.8.8.8), 30 hops max,
    60 byte packets
 1  _gateway (192.168.0.1)  2.275 ms  2.353 ms  2.427 ms
 2  Dinamico-199-198.redeconesul.com.br (186.251.199.198)
    4.895 ms  4.997 ms  4.979 ms
 3  Dinamico-199-197.redeconesul.com.br (186.251.199.197)
    4.958 ms  4.936 ms  4.999 ms
 4  xe-1-3-1.3933.ar4.gru1.gblx.net (64.214.128.61)
    11.088 ms  11.380 ms  11.360 ms
 5  64.209.11.190 (64.209.11.190)  95.183 ms
    ae0-120G.ar4.GRU1.gblx.net (67.16.148.6)
    25.781 ms 64.209.11.190 (64.209.11.190)  94.825 ms
 6  72.14.212.213 (72.14.212.213)
    26.745 ms  23.466 ms  24.700 ms
 7  * * *
 8  google-public-dns-a.google.com (8.8.8.8)
    24.003 ms  24.904 ms  25.148 ms
```

Em resumo, os resultados evidenciam a importância de investigar o impacto das regras e configurações de WAFs nas requisições dos usuários. Dependendo dos requisitos da aplicação, pode ser mais interessante um WAF de menor latência, por exemplo. Por outro lado, um WAF de maior latência, como é o caso do ModSecurity, pode apresentar uma maior acurácia em sua configuração padrão, isto é, sem incluir regras novas e específicas, como pode ser visto na Seção 5.4.

5.4. Eficácia dos WAFs na Segurança de Aplicações Web

Como apresentado e discutido em [Ferrão 2018], a Tabela 5.3 sumariza os resultados obtidos com os testes de três WAFs (ModSecurity, Naxsi e Shadow Daemon) adicionados a cenários controlados contendo uma versão da aplicação Web vulnerável (vide Seção 5.3) implementada nos *frameworks* de desenvolvimento Laravel[3], Synpony[3], Codeigniter[3] e Phalcon[3]. Na tabela constam os nomes dos WAFs, os nomes dos *frameworks*, as vulnerabilidades detectadas pelos respectivos *scanners*, os nomes dos *scanners* e a porcentagem de vulnerabilidades detectadas, respectivamente.

O WAF que apresentou o melhor desempenho foi o ModSecurity, chegando a evitar a detecção de 70% das vulnerabilidades nos *frameworks* Laravel e Symfony, isto é, os *scanners* detectaram apenas 30% das vulnerabilidades existentes na aplicação Web. Os WAFs Shadow Daemon e Naxsi contribuíram menos que o ModSecurity na mitigação da exploração das vulnerabilidades dos ambientes avaliados. No caso do Shadow Daemon, isto era esperado uma vez que este WAF promete mitigar essencialmente vulnerabilidades de injeção de código (como SQL, XML e comandos) e inclusão remota de arquivos. Já o Naxsi afirma mitigar apenas as vulnerabilidades de *SQL injection* e XSS. Portanto, a escolha do WAF pode fazer uma diferença significativa na hora de proteger uma aplicação Web.

Todos os WAFs, em todos os cenários avaliados, mitigaram a vulnerabilidade de injeção de código. No caso do ModSecurity, ele possui o `slr rules`, que é um diretório de arquivos de regras que oferecem proteção contra ataques de *SQL injection*. Os arquivos contêm regras específicas para diferentes assinaturas desta vulnerabilidade. Finalmente, vale ressaltar ainda que nenhum dos WAFs possui regras específicas para evitar as vulnerabilidades de utilização de componentes com vulnerabilidades conhecidas e *Cross Site Requesty Forgery*. Como consequência, estas duas vulnerabilidades não foram mitigadas.

5.5. Considerações Finais

Este minicurso apresentou (a) os conceitos básicos de WAFs *standalone* e SaaS, (b) o estado da arte, (c) aspectos práticos de instalação, configuração e verificação de funcionamento de WAFs *standalone*, (d) o impacto destas ferramentas na latência das requisições Web, e (e) a eficácia na proteção de aplicações Web. Nos itens (c), (d) e (e), foram utilizados como referência WAFs *standalone* gratuitos, como o ModSecurity, Naxsi, Shadow Daemon e xWAF. Enquanto que a instalação e operação destes WAF é relativamente similar, com excessão do xWAF, o impacto na latência das requisições Web varia de forma significativa. No que diz respeito a eficácia em termos de proteção das aplicações Web, novamente, há uma diferença significativa entre os WAFs. Portanto, fica evidente a neces-

Tabela 5.3: Vulnerabilidades detectadas nos cenários dos *frameworks* e WAFs

WAF	*Framework*	Vulnerabilidades detectadas	*Web scanners*	% V.
ModSecurity	Laravel	Má configuração de segurança Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Zed attack proxy] [Nessus]	30%
	Symfony	Exposição de dados sensíveis Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Skipfish] [Nessus]	30%
	Codeigniter	Má configuração de segurança Exposição de dados sensíveis Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Andiparos, Grabber, Paros, Skipfish] [Uniscan, Skipfish, Vega] [Nessus]	40%
	Phalcon	*Cross-site scripting*, Vulnerabilidades conhecidas, *Cross-site requesty forgery* Má configuração de segurança Exposição de dados sensíveis	[Nessus] [Paros] [Vega]	50%
Shadow D.	Laravel	Má configuração de segurança *Cross-site scripting* Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Zed attack proxy] [Nessus, Grabber, Vega, Zed attack proxy] [Nessus]	40%
	Symfony	*Cross-site scripting*, Vulnerabilidades conhecidas, *Cross-site requesty forgery* Má configuração de segurança Exposição de dados sensíveis	[Vega, Nessus] [Andiparos, Paros, Skipfish] [Skipfish]	50%
	Codeigniter	Má configuração de segurança *Cross-site scripting* Exposição de dados sensíveis Quebra de sessão Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Andiparos, Nessus, Paros] [Paros, Vega] [Uniscan, Skipfish, Vega] [Skipfish], Quebra de controle de acesso [Nessus]	70%
	Phalcon	Má configuração de segurança Exposição de dados sensíveis Quebra de sessão, Quebra de controle de acesso Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Paros] [Vega] [Skipfish] [Nessus]	60%
Naxsi	Laravel	*Cross-site scripting* Quebra de controle de acesso Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Nessus, Graber, Vega] [Skipfish] [Nessus]	40%
	Symfony	*Cross-site scripting* Má configuração de segurança Exposição de dados sensíveis Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Vega, Zed attack proxy] [Andiparos, Paros, Skipfish] [Skipfish] [Nessus]	50%
	Codeigniter	Má configuração de segurança Exposição de dados sensíveis Quebra de sessão, Quebra de controle de acesso Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Andiparos, Grabberm Paros] [Uniscan, Skipfish, Vega] [Skipfish] [Nessus]	60%
	Phalcon	*Cross-site scripting* Má configuração de segurança Exposição de dados sensíveis Quebra de sessão Quebra de controle de acesso Vulnerabilidades conhecidas, *Cross-site requesty forgery*	[Vega, Zed attack proxy] [Paros] [Vega] [Skipfish] [Skipfish] [Nessus]	70%

sidade de investigação e avaliação dos WAFs para verificar o atendimento aos requisitos específicos da aplicação Web. Enquanto um WAF como o xWAF pode ser adequado e interessante para uma aplicação Web PHP, devido ao seu baixo impacto na latência das requisições destinadas à aplicação, ele certamente não é um WAF adequado para as demais aplicações Web, desenvolvidas nas mais variadas linguagens de programação, como JavaScript, Java, Python, Perl, C e Go.

Referências

[Acunetix 2019] Acunetix (2019). Web Application Vulnerability Report. https://bit.ly/2wh62TH.

[Cimpanu 2019] Cimpanu, C. (2019). Security bug would have allowed hackers access to Google's internal network. https://zd.net/2F8Mju8.

[Clincy and Shahriar 2018] Clincy, V. and Shahriar, H. (2018). Web Application Firewall: Network Security Models and Configuration. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 01, pages 835–836.

[Convergência Digital 2016] Convergência Digital (2016). Ataques hackers provocaram um prejuízo de R$ 30 bilhões no Brasil. http://bit.do/hackers-prejuizo.

[Ferrão 2018] Ferrão, I. G. (2018). Análise black-box de ferramentas de segurança na Web. Trabalho de conlusão de curso, Curso de Ciência da Computação, Universidade Federal Do Pampa. http://dspace.unipampa.edu.br:8080/jspui/handle/riu/3695.

[Ferrão et al. 2018] Ferrão, I. G., de Macedo, D. D. J., and Kreutz, D. (2018). Investigação o do Impacto de Frameworks de Desenvolvimentode Software na Segurança de Sistemas Web. In *3o Workshop Regional de Segurança da Informação e de Sistemas Computacionais (WRSeg)*. http://arxiv.kreutz.xyz/wrseg2018_scanners_frameworks.pdf.

[Funk et al. 2018] Funk, R., Epp, N., and A., C. C. (2018). Anomaly-based Web Application Firewall using HTTP-specific features and One-Class SVM. *Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação*, 2(1).

[HackerOne 2019] HackerOne (2019). The HackerOne Top 10 Most Impactful and Rewarded Vulnerability Types. https://bit.ly/2RiWbGu.

[Machado et al. 2019] Machado, R., Kreutz, D., Paz, G., and Rodrigues, G. (2019). Vazamentos de Dados: Histórico, Impacto Socioeconômico e as Novas Leis de Proteção de Dados. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*. http://errc.sbc.org.br/2019/wrseg/papers/machado2019vazamentos.pdf.

[Melchior et al. 2019] Melchior, F., Kreutz, D., and Fiorenza, M. (2019). Web Application Firewalls (WAFs): o impacto do número de regras na latência das requisições Web. In *4o Workshop Regional de Segurança da Informação e de Sistemas Computacionais*. http://errc.sbc.org.br/2019/wrseg/papers/melchior2019web.pdf.

[Moosa and Alsaffar 2008] Moosa, A. and Alsaffar, E. M. (2008). Proposing a Hybrid-intelligent Framework to Secure e-Government Web Applications. In *Proceedings of the 2Nd International Conference on Theory and Practice of Electronic Governance*, ICEGOV '08, pages 52–59, New York, NY, USA. ACM.

[Portinari 2018] Portinari, N. (2018). Ataques hackers são mais temidos por empresas que inflação e austeridade. http://bit.do/ataque-hacker.

[Rao et al. 2016] Rao, G. R. K., Prasad, R. S., and Ramesh, M. (2016). Neutralizing Cross-Site Scripting Attacks Using Open Source Technologies. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, ICTCS '16, pages 24:1–24:6, New York, NY, USA. ACM.

[Razzaq et al. 2013] Razzaq, A., Hur, A., Shahbaz, S., Masood, M., and Ahmad, H. F. (2013). Critical analysis on web application firewall solutions. In *2013 IEEE Eleventh*

International Symposium on Autonomous Decentralized Systems (ISADS), pages 1–6.

[Rietz et al. 2016] Rietz, R., König, H., Ullrich, S., and Stritter, B. (2016). Firewalls for the Web 2.0. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 242–253.

[Romani 2016] Romani, B. (2016). : 6 casos de ataque hacker. https://super.abril.com.br/tecnologia/6-casos-de-ataque-hacker/.

[Singh et al. 2018] Singh, J. J., Samuel, H., and Zavarsky, P. (2018). Impact of Paranoia Levels on the Effectiveness of the ModSecurity Web Application Firewall. In *2018 1st International Conference on Data Intelligence and Security (ICDIS)*, pages 141–144.

[Srokosz et al. 2018] Srokosz, M., Rusinek, D., and Ksiezopolski, B. (2018). A New WAF-Based Architecture for Protecting Web Applications Against CSRF Attacks in Malicious Environment. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 391–395.