Organizadores:

Danielo G. Gomes (UFC)
Igor M. Moraes (UFF)
Miguel Elias M. Campista (UFRJ)

SBRC 2020

# Minicursos do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

Sociedade Brasileira de Computação

Porto Alegre
2020


**Organizadores:**

Danielo G. Gomes (UFC)
Igor M. Moraes (UFF)
Miguel Elias M. Campista (UFRJ)

# Minicursos do XXXVIII
## Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos

Sociedade Brasileira de Computação

Porto Alegre
2020

Dados Internacionais de Catalogação na Publicação (CIP)

S612 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (38. : 2020 : Porto Alegre, RS)

Anais do 38ª Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos [recurso eletrônico] – Organizadores: Danielo G. Gomes, Igor M. Moraes, Miguel Elias M. Campista – Porto Alegre : SBC, 2021.

ISBN 978-65-87003-33-7

1. Computação. 2. Rede de computadores. 3. Sistemas distribuídos. I. Gomes, Danielo G. II. Moraes, Igor M. III. Campista, Elias M. IV. Sociedade Brasileira de Computação. V. Laboratório Nacional de Redes de Computadores. VI. Título.

CDU 004

Catalogação elaborada por Francine Conde Cabral
CRB-10/2606

# Sociedade Brasileira de Computação – SBC



**Contato**
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS

CNPJ: 29.532.264/0001-78
http://www.sbrc.org.br

# Laboratório Nacional de Redes de Computadores (LARC)



**Contato**
Universidade Federal do Ceará
GREat
Av. Mister Hull, Bloco 942-A - Campus do Pici,
Fortaleza - CE - Brasil
CEP: 60440-554

CNPJ: 29.532.264/0001-78
http://www.sbrc.org.br

# Organização do SBRC 2020

**Coordenadores Gerais**
Igor M. Moraes (UFF)
Miguel Elias M. Campista (UFRJ)

**Coordenadores do Comitê de Programa**
Marcelo G. Rubinstein (UERJ)
Anelise M. Fonseca (UTFPR)

**Coordenador do Workshop**
Célio Vinícius N. de Albuquerque (UFF)

**Coordenador de Palestras, Tutoriais e Mentoria**
Marcelo Dias de Amorim (CNRS)

**Coordenadora de Painéis**
Jussara M. Almeida (UFMG)

**Coordenador de Minicursos**
Danielo G. Gomes (UFC)

**Coordenador do Salão de Ferramentas**
Rodrigo S. Couto (UFRJ)

**Coordenador do Concurso de Teses e Dissertações**
Ronaldo A. Ferreira (UFMS)

**Coordenadores do Hackathon**
Alex B. Vieira (UFJF)
Michelle S. Wangham (Univali)

**Comitê de Organização Local**
Dianne Scherly V. Medeiros (UFF)
Diogo M. F. Mattos (UFF)
Diego G. Passos (UFF)
Marcel W. R. da Silva (UFRRJ)

**Comitê Consultivo**
Alberto Egon Schaeffer Filho
Ronaldo A. Ferreira (UFMS)
Weverton Cordeiro (UFRGS)
Fábio Luciano Verdi (UFSCAR)
Jó Ueyama (USP)
Fabíola Gonçalves Pereira Greve (UFBA)
Antônio Jorge Gomes Abelém (UFPA)
Rossana Andrade (UFC)
Luiz Fernando Bittencourt (Unicamp)

# Sinopse

O livro Minicursos do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos contém os minicursos selecionados para apresentação no XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), realizado online entre os dias 7 e 10 de dezembro de 2020. O Livro dos Minicursos do SBRC tem sido tradicionalmente utilizado como material de estudo de alta qualidade por alunos de graduação e pós-graduação, bem como por profissionais da área. As sessões de apresentações dos minicursos são também uma importante oportunidade para atualização de conhecimentos da comunidade científica e para complementação da formação dos participantes. O principal objetivo dos Minicursos do SBRC é oferecer treinamento e atualização de curto prazo em temas normalmente não cobertos nas estruturas curriculares e que possuem grande interesse entre acadêmicos e profissionais.

# Synopsis

The book Short Courses of the 38th Symposium on Computer Networks and Distributed Systems comprises the short courses selected for presentation at the 38th Brazilian Symposium on Computer Networks and Distributed Systems (SBRC), held online between December 7 and 10, 2020. The SBRC Short Courses Book has traditionally been used as high quality study material by undergraduate and graduate students as well as by IT professionals who work on computer networking and distributed systems. The short-courses presentations sessions are also an important opportunity to update the knowledge of the scientific community and to complement the attendees training. The main objective of the SBRC Short Courses is to offer short-term training and updating on topics not normally covered in the curriculum and to make both the students and professionals more interested in the area.

# Mensagem dos Coordenadores Gerais do SBRC 2020

É com enorme satisfação que acolhemos todos no XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), realizado entre os dias 7 e 10 de dezembro de 2020. O SBRC 2020 será o primeiro evento da série realizado totalmente online, como consequência da pandemia provocada pelo coronavírus. Os desafios foram muitos, exigindo o replanejamento completo do evento, o que teve consequências tanto na data de realização, prevista para ocorrer entre 25 e 29 de maio de 2020, até a modalidade, que deixou de ser presencial no Rio de Janeiro capital para ser online. Apesar de todas as dificuldades, é um grande privilégio organizar um evento de tamanha envergadura, capaz de reunir uma comunidade repleta de profissionais com reconhecido destaque nacional e internacional. A contribuição científica e a manutenção da qualidade técnica dos trabalhos tem como resultado o fortalecimento do Brasil como país pioneiro em pesquisa, desenvolvimento e inovação. Contribuição esta que não esmoreceu, mesmo diante do cenário adverso atravessado. Como consequência, o SBRC 2020 trouxe uma programação rica, de ponta, organizada após um cuidadoso processo de seleção. Ao final, foram selecionados 72 artigos completos, cujas apresentações foram distribuídas em 18 sessões técnicas, 9 ferramentas para apresentação no Salão de Ferramentas, 5 minicursos e 16 trabalhos candidatos a prêmios de melhor tese e dissertação nas áreas de interesse do evento no último ano. A programação do SBRC 2020 conta também com 5 palestras proferidas por pesquisadores renomados internacionalmente, 3 painéis de discussões e debates, bem como 8 workshops, o evento MUSAS e o Hackathon. Além da programação técnica, o SBRC também rende homenagens através do prêmio "Destaque do SBRC", que anualmente reconhece a contribuição de um pesquisador de destaque da área. A organização do SBRC 2020 gostaria de agradecer o apoio incondicional da SBC, do LARC, do Comitê Consultivo do SBRC e da Comissão Especial de Redes de Computadores e Sistemas Distribuídos da SBC, sem os quais não seria possível realizar o evento. Estejam cientes de que a ajuda prestada, em especial nos momentos críticos de decisão sobre o evento, foi determinante para alcançarmos sucesso. Agradecemos também o importante apoio financeiro do Comitê Gestor da Internet no Brasil (CGI.br), do CNPq, da CAPES, da FAPERJ, do Instituto de Computação da UFF, da Escola Politécnica e do Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia (COPPE) da UFRJ e da Google. Nosso especial agradecimento à Universidade Federal Fluminense (UFF) e à Universidade Federal do Rio de Janeiro (UFRJ) pela oportunidade e pelo indispensável suporte à realização do evento. Nossos carinhosos agradecimentos aos competentes e incansáveis coordenadores do comitê de programa, Anelise Munaretto (UTFPR) e Marcelo G. Rubinstein (UERJ); ao coordenador de minicursos, Danielo G. Gomes (UFC); ao coordenador dos workshops, Célio Vinícius N. de Albuquerque (UFF); ao coordenador do concurso de teses e dissertações, Ronaldo A. Ferreira (UFMS); à coordenadora de painéis e debates, Jussara M. Almeida (UFMG); ao coordenador do Salão de Ferramentas, Rodrigo S. Couto (UFRJ); ao coordenador de palestras e tutoriais, Marcelo Dias de Amorim (CNRS); e aos coordenadores do Hackathon, Michelle S. Wangham (Univali) e Alex B. Vieira (UFJF). Um agradecimento especial ao comitê local coordenado pela professora Dianne Scherly V. Medeiros (UFF) e pelos professores Diogo M. F. Mattos (UFF), Diego G. Passos (UFF) e Marcel W. R. da Silva (UFRRJ) que não envidaram esforços em conjunto com toda a equipe local liderada pelo aluno Kaylani Bochie (UFRJ), nosso responsável web. Por fim, agradecemos ao setor gráfico da COPPE pela realização das artes e o apoio da Creacteve.

Igor M. Moraes (UFF)
Miguel Elias M. Campista (UFRJ)
Coordenadores Gerais do SBRC 2020

# Mensagem do Coordenador de Minicursos do SBRC 2020

É com muita satisfação que apresento os Minicursos do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), realizado online entre os dias 7 e 10 de dezembro de 2020. Os Minicursos do SBRC produzem os capítulos do Livro de Minicursos os quais permitem à comunidade se atualizar em temas que despertam grande interesse entre acadêmicos e profissionais. O livro, assim como as apresentações no simpósio, tem sempre um viés altamente didático, capaz de motivar alunos de graduação, pós-graduação e profissionais da área a mergulharem nos temas propostos. A ideia é abrir horizontes e complementar a formação dos participantes em temas recentes ou ainda pouco explorados nas estruturas curriculares das instituições tradicionais de ensino e pesquisa e com grande potencial de interesse para estudantes e profissionais de forma a melhor qualificá-los para suas pesquisas acadêmicas e/ou sua atuação no mercado. Para este ano, recebemos 21 registros/submissões no prazo final após dois adiamentos. Estes adiamentos ocorreram na perspectiva de atrair mais submissões, como de fato ocorreu: de 11 propostas em 29/11/2019 (prazo original), passamos a contar com 16 propostas em 6/12/2019 (1o adiamento) e com 21 propostas em 13/12/2019 (prazo final). Uma das 21 propostas foi removida a pedido dos autores, de modo que tivemos ao todo 20 propostas revisadas. Todas as 20 propostas foram atribuídas a três revisores, os quais aceitaram prontamente e revisaram no prazo. Houve poucos atrasos nas revisões, demonstrando profundo comprometimento com o evento mesmo em época de férias. Por isso, também gostaria de agradecer imensamente aos 30 membros do TPC deste ano que se dedicaram à revisão das propostas de forma sempre atenciosa. Foi um prazer trabalhar com todos. Dos 20 registros/submissões de propostas decidiu-se, para evitar fracionamento de público no evento, aceitar no máximo cinco propostas. Esta taxa de corte de 25% está conforme a adotada na trilha principal do SBRC e, portanto, foi julgada razoável. As cinco propostas selecionadas, aquelas que tiveram as maiores médias gerais, são certamente ótimas candidatas a atrair público ao evento. Os temas selecionados são bastante atuais e abordam fog computing, quantum Internet, softwarização 5G, deep learning e computação serverless. Vale mencionar que, dentre as propostas não contempladas, algumas teriam plenas condições de serem aceitas, o que infelizmente não foi possível desta vez. Gostaria de agradecer aos organizadores gerais do SBRC, os professores Miguel Elias M. Campista e Igor M. Moraes os quais, com muita dedicação, otimismo e resiliência, não desistiram de organizar o SBRC neste ano atípico de 2020. Agradeço-os especialmente pela confiança na organização dos Minicursos, no trato sempre muito gentil e na infinita paciência. Todos as mensagens enviadas foram sempre respondidas prontamente. Agradeço ainda aos professores Dianne Scherly V. Medeiros, Diogo M. F. Mattos, Diego G. Passos e Marcel W. R. da Silva, membros do comitê local de organização. Agradeço também aos autores dos minicursos, pois sem a dedicação e esmero de todos na escrita e na entrega pontual do material, nada disso teria sido viável. Tenho certeza que todas as apresentações brindaram todo o trabalho e possibilitaram plena absorção técnica da plateia. O desfecho dos minicursos foi suave graças à colaboração e ao profissionalismo de todos. Agora, é rentabilizar o árduo trabalho fomentando frutíferas discussões em direção ao avanço da Ciência, cada vez mais útil e necessária nestes dias.

Danielo G. Gomes
Coordenador de Minicursos do SBRC 2020

# Comitê de Programa dos Minicursos do SBRC 2020

Alberto Schaeffer-Filho (UFRGS)
Alex Vieira (UFMG)
Alfredo Goldman (USP)
Artur Ziviani (LNCC)
Christian Esteve Rothenberg (UNICAMP)
Daniel de Oliveira (UFF)
Daniel Fernandes Macedo (UFMG)
Daniel Guidoni (UFSJ)
Daniel Menasché (UFRJ)
Dianne Medeiros (UFF)
Edmundo Madeira (UNICAMP)
Eduardo Cerqueira (UFPA)
Fabíola Greve (UFBA)
Giovanni Comarela (UFV)
Gustavo Figueiredo (UFBA)
Heitor Ramos (UFMG)
Igor Moraes (UFF)
José Augusto Suruagy Monteiro (UFPE)
Kelvin Dias (UFPE)
Leobino Sampaio (UFBA)
Luis Carlos De Bona (UFPR)
Luiz Fernando Bittencourt (UNICAMP)
Marcelo Dias de Amorim (Sorbonne Université – CNRS)
Michele Nogueira (UFPR)
Rafael Lopes Gomes (UECE)
Raquel S. Cabral (UFAL)
Rodrigo de Souza Couto (UFRJ)
Ronaldo Ferreira (UFMS)
Rossana Andrade (UFC)
Thiago Henrique Silva (UTFPR)

# Sumário

# Capítulo 1

# Plataformas de *Fog Computing*: da Teoria à Prática

Thiago P. Silva (UFMT), Aluizio Rocha (UFRN), Thais Batista (UFRN), Frederico Lopes (UFRN), Flávia C. Delicato (UFF), Paulo F. Pires (UFF)

***Abstract***

*Nowadays we are witnessing a significant increase in the amount of IoT devices and the volume of data generated by these devices. Associated with this, more IoT applications have ultra low latency and response time requirements, such that cloud-centric IoT platforms are unable to address these requirements and to adequately handle the increasing volume of data. To support these new applications and handle the increasing volume of IoT data, Fog Computing platforms have been proposed. The Fog Computing paradigm is characterized by a horizontal, system-level architecture where devices that are in the path of IoT data, from its origin in IoT devices to the cloud, are used for processing distribution, storage, and data control. Fog Computing platforms aim to abstract the highly dynamic and heterogeneous environment where they are inserted, in order to facilitate the development of applications by users. This chapter has the following objectives: (i) to present the Fog Computing paradigm; (ii) to survey and discuss the main requirements that guide the development of Fog Computing platforms; (iii) to discuss two proposals of reference architecture for Fog Computing; (iv) to describe the main Fog Computing platforms and contrast them with the requirements previously raised; (v) to demonstrate in a practical way the use of a Fog Computing platform for the context of smart cities.*

***Resumo***

*Atualmente presenciamos um aumento significativo na quantidade de dispositivos IoT e no volume de dados gerados por estes dispositivos. Associado a isto, cada vez mais aplicações IoT possuem requisitos de latência e tempo de resposta ultra baixo, de tal forma que as plataformas IoT centradas na nuvem não conseguem endereçar estes requisitos e tão pouco tratar de maneira adequada o volume crescente de dados. Para suportar estas novas aplicações e tratar o volume crescente de dados IoT, plataformas que empregam*

*o paradigma Fog Computing têm sido propostas. O paradigma Fog Computing é caracterizado por uma arquitetura horizontal em nível de sistema onde os dispositivos que estão no caminho percorrido pelos dados IoT, desde sua origem nos dispositivos IoT até a nuvem, são utilizados para distribuição de processamento, armazenamento e controle de dados. As plataformas que incorporam o paradigma Fog Computing visam abstrair o ambiente altamente dinâmico e heterogêneo onde estão inseridas, de forma a facilitar o desenvolvimento de aplicações. Este capítulo tem como objetivos: (i) apresentar o paradigma Fog Computing; (ii) levantar e discutir os principais requisitos que norteiam o desenvolvimento das plataformas Fog Computing; (iii) discutir duas propostas de arquitetura de referência para Fog Computing; (iv) descrever as principais plataformas Fog Computing e contrapô-las aos requisitos levantados; (v) demonstrar, de forma prática, o uso de uma plataforma Fog Computing para o contexto de cidades inteligentes.*

## 1.1. Introdução

A Internet das Coisas (do inglês *Internet of Things – IoT*) [1] é um paradigma de computação que engloba *software*, *hardware* e serviços visando interconectar objetos físicos com a infraestrutura de comunicação da Internet. Tais objetos, comumente chamados de dispositivos IoT, são dotados de sensores e atuadores e possuem, em sua maioria, baixo poder de processamento e armazenamento. Entretanto, quando estes objetos são interconectados, eles moldam uma rede de dispositivos inteligentes capaz de realizar vários tipos de processamentos, capturar variáveis ambientais e reagir a estímulos externos, fornecendo serviços de valor agregado ao usuário. Atualmente existem diferentes fabricantes de dispositivos IoT e uma infinidade de sensores e atuadores, que possuem diferentes capacidades de monitorar e controlar remotamente o ambiente em que estão inseridos. Desta forma, a interconexão dos dispositivos IoT, associada ao barateamento da tecnologia e a sua capacidade de promover inovação, impulsionam a criação de aplicações interessantes em vários domínios.

Dentre os diversos domínios possíveis de uso da tecnologia IoT, destaca-se o domínio das cidades inteligentes, no qual o uso de tecnologias avançadas de comunicação e sensoriamento visa prover serviços de valor agregado para os órgãos administrativos das cidades e para seus cidadãos [2]. Ainda no domínio de cidade inteligente, a tecnologia IoT, conforme aponta [3], forma a infraestrutura de sensores necessária para que as cidades melhorem sua eficiência operacional e forneçam serviços de qualidade, os quais impactam positivamente no bem-estar dos cidadãos. Entretanto, a inerente heterogeneidade dos ambientes de IoT requer soluções que permitam a interoperabilidade e integração dos diversos componentes que fazem parte desse ambiente. Neste sentido, naturalmente plataformas de *middleware* surgiram como solução para prover interoperabilidade e gerenciar a crescente variedade de dispositivos associados a aplicações e o consumo de dados por parte dos usuários finais [4, 5]. A maioria das plataformas para IoT são centradas na nuvem e utilizam o paradigma *Cloud of Things* (CoT), que preconiza a integração sinergética entre IoT e a nuvem. Em sistemas de CoT, tipicamente os dados gerados por dispositivos IoT fluem continuamente através da infraestrutura de comunicação da Internet até chegar à nuvem, onde são processados e armazenados [6].

Atualmente presenciamos um aumento significativo na quantidade de dispositivos IoT, de modo que, em um futuro próximo, o volume de dados gerados por eles não

poderá ser tratado adequadamente por plataformas centradas na nuvem. Além disso, conforme [7, 8] argumentam, soluções IoT centralizadas na nuvem não apresentam modelos econômicos e sustentáveis, uma vez que o custo de largura de banda e armazenamento pode ser substancialmente alto ao enviar todos os dados IoT para a nuvem. Ademais aos problemas supracitados, a tecnologia IoT impulsionou o surgimento de aplicações e serviços que requerem latência e tempo de resposta ultra baixo e, portanto, plataformas centradas na nuvem nem sempre podem suprir as necessidades deste tipo de aplicação. Por exemplo, algumas aplicações em que milissegundos têm um impacto significativo em sua execução, como na comunicação veículo-veículo, para evitar colisões ou acidentes, não são adequadas para uma arquitetura centralizada como a proposta da CoT [9].

Os novos desafios impostos pelas aplicações emergentes mobilizaram a academia e indústria a discutirem soluções viáveis para lidar com tais desafios e atingir o pleno potencial da IoT. Uma nova tendência atualmente é migrar a computação - seja ela na forma de processamento ou armazenamento - da camada de nuvem para as camadas próximas aos dispositivos IoT. Esse novo paradigma é chamado de *fog computing* e é caracterizado por uma arquitetura horizontal em nível de sistema que contrasta com o modelo CoT [10]. No paradigma *fog computing*, os dispositivos que estão no caminho (chamado de *Cloud-to-Thing Continuum*) percorrido pelos dados IoT, desde sua origem nos dispositivos finais IoT até a nuvem, são utilizados para distribuição de processamento, armazenamento e controle de dados [8]. Os dispositivos que estão no *Cloud-to-Thing Continuum* são chamados de nós *fog* (do inglês, *fog node*) e incluem roteadores, pontos de acesso, *gateways*, entre outros dispositivos que, atualmente, possuem maior capacidade de armazenamento e processamento computacional que seus antecessores.

O desenvolvimento de plataformas que incorporam o paradigma de *fog computing* é uma área de pesquisa recente e em evolução. Isto posto, o paradigma apresenta muitos desafios não superados e várias ideias ainda estão em processo de elaboração e desenvolvimento [11, 3, 12]. Como consequência do grau de incipiência da tecnologia, a maioria dos trabalhos publicados na literatura que abordam o tema *fog computing* são de cunho teórico. Porém, o consórcio OpenFog, formado por várias universidades e grandes empresas da área de comunicação e tecnologia, numa tentativa de progresso da adoção e padronização desse paradigma, criou uma Arquitetura de Referência (AR) para *fog computing*. Tal arquitetura é formada por oito pilares que representam as principais características de um sistema que visa fornecer a distribuição das funções de computação, armazenamento, controle e rede mais próximas à fonte de dados ao longo do *Cloud-to-Thing Continuum*. Em julho de 2018, o IEEE adotou a arquitetura de referência como o primeiro padrão para *fog computing* [13].

Apesar dos esforços do consórcio OpenFog e IEEE, observa-se que as plataformas existentes apresentam características distintas e muitas vezes não alinhadas aos pilares estabelecidos pela arquitetura de referência, desta forma não contribuindo para a interoperabilidade e formando silos isolados. Associado a isto, ainda não existe um consenso a respeito da definição do termo *fog computing* e, por vezes, o termo é usado de forma intercambiável com *Edge Computing*. Considerando a relevância do paradigma e o potencial de aplicabilidade no contexto de cidades inteligentes, de forma a criar plataformas disruptivas, este capítulo tem como objetivos: i) apresentar os principais conceitos do paradigma bem como os requisitos de *software* que direcionam o projeto de plataformas

*fog computing*; ii) apresentar e discutir as principais arquiteturas de referência publicadas na literatura; iii) apresentar quatro plataformas *fog computing* com ênfase em cidades inteligentes; iv) demonstrar um estudo de caso implementado em um plataforma alvo.

Neste capítulo, para facilitar o entendimento a respeito do paradigma, adotaremos a definição proposta pelo modelo conceitual de *fog computing* criado pelo NIST [10]. Tal definição está alinhada ao propósito do OpenFog e considera *edge computing* apenas como a camada que compreende os dispositivos finais e usuários, tipicamente chamada de borda da rede (do inglês, *edge network)*. Já *fog computing* oferece uma arquitetura multi-camadas organizada de forma hierárquica para a execução de aplicações no *Cloud-to-Thing Continuum*. Sendo assim, segundo essa visão *edge computing* limita-se a execução de aplicações específicas em uma rede fixa formada por um conjunto limitado de dispositivos acessíveis de forma direta que estão na mesma rede, desconsiderando a nuvem. Por outro lado, *fog computing* utiliza, inclusive, os dispositivos da borda da rede, mas engloba esse continuum desde os dispositivos produtores de dados até a nuvem.

Outra definição importante adotada neste texto refere-se a distinção entre dispositivos IoT e os dispositivos nós *fog*. Consideramos o dispositivo IoT um *hardware* que tem sensores e atuadores a ele acoplado, tendo as capacidades de sensoriamento e atuação no ambiente em que está inserido. Tipicamente, os dispositivos IoT são os produtores de dados enquanto os nós *fog* são organizados e interconectados formando uma rede de nós *fog* (do inglês *fog network*) para processar e/ou armazenar estes dados. Neste sentido, a Internet das Coisas proporciona a interconexão dos dispositivos IoT entre e (direta ou indiretamente) com a Internet, permitindo a criação de serviços/aplicações de valor agregado [14]. Por outro lado, um nó *fog* é um dispositivo que tem capacidades de computação, armazenamento e rede, essenciais para a execução de aplicações IoT [15]. Apesar do fato de que atualmente dispositivos IoT como, por exemplo, Raspberry e BeagleBone, também podem atuar como nós *fog*, ao longo do texto deixaremos claro o papel que estes dispositivos exercem, de forma a não confundir o leitor.

O restante deste capítulo está estruturado da seguinte forma: a Seção 1.2 apresenta os principais requisitos que direcionam o desenvolvimento de plataformas de *fog computing*; a Seção 1.3 discute as principais propostas de arquitetura de referência para área; a Seção 1.4 apresenta as plataformas *fog computing* de código aberto e uma comparação entre elas é proposta; na Seção 1.5 será apresentado um estudo de caso implementado na plataforma FogFlow para a detecção e identificação facial utilizando-se processamento de vídeo em dispositivos de baixo custo; e por fim, a Seção 1.6 apresenta as conclusões e perspectivas futuras.

## 1.2. Requisitos de plataformas de *fog computing*

Esta seção apresenta os principais requisitos que direcionam a construção de plataformas de *fog computing*. Tais requisitos visam atender as necessidades das aplicações e dos usuários, e foram levantados a partir da análise dos principais trabalhos publicados na área [16, 17, 18, 19, 20, 21, 22, 9, 12, 18, 23, 24]. Cabe ressaltar que o conjunto de requisitos que serão apresentados determinam o que uma plataforma *fog computing* de propósito geral necessita prover/atender. Entretanto, o que observamos na prática, é que as plataformas atuais são em sua maioria específicas de domínio e, por conseguinte,

possuem diferentes características e atendem também a requisitos inerentes ao domínio onde estão inseridas. Os seguintes requisitos serão aqui apresentados: 1) Distribuição Geográfica; 2) Suporte a aplicações sensíveis a latência; 3) Suporte a Heterogeneidade; 4) Interoperabilidade; 5) Otimização do uso da largura de banda; 6) Ciência de Localização; 7) Ciência de Contexto; 8) Suporte a Mobilidade; 9) Disponibilidade; 10) Eficiência Energética; 11) Segurança e Privacidade; 12) Escalabilidade.

O requisito de **distribuição geográfica** diz respeito à distribuição em diferentes localizações geográficas dos nós *fog* que a plataforma necessita gerenciar para fornecer a distribuição das funções de computação, armazenamento, controle e rede mais próximas à fonte de dados e/ou usuários. No paradigma *fog computing*, em contraste com a computação em nuvem tradicional, os recursos e serviços estão distribuídos em posições estratégicas em uma área geográfica potencialmente ampla, para atender a uma grande variedade de usuários [18]. Desta forma, segundo [25], as plataformas precisam escolher de forma adequada os nós *fog* que executarão as aplicações, levando em consideração não somente as posições geográficas, mas também os critérios de QoS (Qualidade de Serviço, do inglês *Quality of Service*) demandados pela aplicação. Tal escolha precisa ser transparente para as aplicações e usuários, i.e., os usuários não precisam saber da localização onde suas aplicações serão executadas.

Um dos principais objetivos de *fog computing* é reduzir a latência, de forma a **habilitar aplicações ultra sensíveis à latência**. Este tipo de aplicação requer baixa latência fim-a-fim e, portanto, os componentes ou unidades de processamentos que compõem tais aplicações não podem ser completamente executados na nuvem. Por exemplo, jogos de realidade alternativa (do inglês, *reality games*) requerem latência ultra baixa, menores que 20 ms, conforme aponta [18]. Outro exemplo, no contexto de veículos conectados (do inglês *connected vehicles*), são as aplicações de segurança como, por exemplo, aviso de violação de sinal de trânsito, aviso colaborativo de colisão frontal e aviso de mudança de faixa de rolagem. Tais aplicações requerem latência menor que 100 ms, conforme aponta [26]. Desta forma, nas aplicações exemplificadas, os dados oriundos de dispositivos finais (aqueles localizados na borda da rede) precisam alcançar o local onde serão processados e/ou armazenados, e uma resposta deverá ser dada em um intervalo de tempo ultra baixo. Ao considerarmos a computação em nuvem, normalmente a latência entre o usuário final e a nuvem é em torno de 20 a 40 ms em redes cabeadas, atingindo 150 ms em redes 4G. Além da latência na comunicação, deve-se considerar também o problema de congestionamento da rede que é mais comum em soluções centradas na nuvem, pois a nuvem está a vários saltos (*hops*) de distância dos dispositivos produtores de dados e o dados precisam percorrer diferentes enlaces que são compartilhados por diversos usuários e aplicações.

Apenas a distribuição geográfica dos nós *fog* não é suficiente para assegurar baixa latência, pois uma característica importante de *fog computing* é usar nós *fog* heterogêneos, que possuem recursos variados e são interligados por diferentes meios de comunicação (cabo, rede sem fio, etc). Portanto, as plataformas também devem considerar a velocidade dos links que interligam os nós *fog*, de forma a distribuir o processamento e/ou armazenamento nos nós das camadas mais próximas ao usuário final e/ou fonte de dados, considerando o link de comunicação entre eles. Além dos nós *fog* heterogêneos, a plataforma também precisa lidar com diferentes tipos de dispositivos IoT, com seus sensores e atuadores que apresentam capacidades, protocolos de comunicação e representação de

dados distintas. Portanto, o requisito de **suporte a heterogeneidade** diz respeito à capacidade da plataforma de assegurar o devido gerenciamento e coordenação da rede dos dispositivos IoT, bem como os nós *fog* heterogêneos, de modo a oferecer uma abstração apropriada para ocultar os detalhes de heterogeneidade dos recursos. Tal requisito é essencial para evitar que os desenvolvedores de aplicações tenham que lidar diretamente com os detalhes técnicos de cada dispositivo IoT e nós *fog* que executarão as aplicações.

É vital garantir que os componentes de uma plataforma *fog computing* forneçam **interoperabilidade** para liberar o potencial da IoT. Conforme discute [14], as redes IoT são largamente distribuídas, heterogêneas e oferecem vários tipos de serviços (do inglês *multi-services*). Consequentemente, o risco de não-interoperabilidade é alto, levando a indisponibilidade de alguns serviços para os usuários/aplicações finais ou a perda de informações importantes fora da IoT devido a falta de interoperabilidade. Ainda segundo [14], existem alguns desafios para interoperabilidade na IoT: i) a coexistência de muitos dispositivos no ambiente que precisam se comunicar e trocar informações; ii) muitos dispositivos são concebidos por diferentes fabricantes para diferentes propósitos, visando diversos domínios de aplicação; iii) a dinamicidade do ambiente, pois dispositivos que suportam novas capacidades e utilizam protocolos imprevistos, mas que precisam se comunicar e compartilhar dados na IoT, podem entrar e sair do ambiente; iv) a existência de muitos formatos de dados que necessitam seguir os mesmo princípios de modelagem e podem ser inter-relacionados. A interoperabilidade é, muitas vezes, centrada em protocolos de comunicação, tipicamente M2M (do inglês *Machine-to-Machine*), e na infraestrutura necessária para que esses protocolos funcionem. Desta forma, os componentes de *hardware/software* e serviços de uma plataforma *fog computing* precisam implementar os protocolos desenvolvidos por órgãos de padronização como, por exemplo IEEE e ISO, de forma a promover a comunicação entre os dispositivos e uniformizar os procedimentos de conectividade e troca de dados, promovendo a interoperabilidade. Conforme aponta o consórcio OpenFog [13], a interoperabilidade é essencial para o sucesso de um ecossistema formado por plataformas *fog computing* e aplicações IoT, evitando o surgimento de silos isolados.

As aplicações que produzem grande volume de dados como, por exemplo, videomonitoramento (do inglês *video surveillance*), tipicamente necessitam de grande largura de banda, e soluções centradas na nuvem requerem que todo o *stream* de vídeo seja enviado para nuvem para que o processamento adequado possa ser realizado. Entretanto, um cenário onde uma câmera digital captura vídeo em alta definição (HD, do inglês *High Definition*) requer no mínimo 1 Mbps de *bitrate* e no mínimo 2 Mbps de taxa de *upload*. Já uma transmissão em Full-HD precisa de no mínimo 3 Mbps de *bitrate* e 6 Mbps de taxa de *upload*[1]. Uma aplicação para videomonitoramento pode ser executada por uma plataforma *fog computing* utilizando-se os dispositivos computacionais da borda da rede que estão próximos à fonte de *streaming* de vídeo. Desta forma, pode-se **otimizar o uso da largura de banda**, na medida que o dado bruto poderá ser tratado na *fog* e, por exemplo, apenas um subconjunto de informações necessitarão ser transmitidas para a nuvem.

Para endereçar o requisito de **ciência de localização**, a plataforma precisa conhecer a localização física e topológica dos dispositivos IoT e dos nós *fog*. Considerando um

[1]Cálculo de bitrate - https://toolstud.io/video/bitrate.php

cenário de cidade inteligente, onde nós *fog* estão espalhados e podem se tornar indisponíveis a qualquer momento, a plataforma precisa fazer uso de estratégias de descoberta e gerenciamento de dispositivos presentes no ambiente. Ademais, a plataforma precisa saber o conjunto de nós *fog* que estão em uma determinada região geográfica, quais serviços e aplicações estão sendo executados nestes nós, bem como as informações das cargas de trabalho e as condições da rede. Por outro lado, a **ciência de contexto** é a capacidade da plataforma de recuperar informações a respeito das mudanças de estado das diferentes variáveis contextuais de seu interesse no ambiente dinâmico em que os seus componentes estão inseridos. O contexto é qualquer informação que pode ser usada para caracterizar o estado de uma entidade [27]. Entidades podem ser, por exemplo, uma pessoa, uma aplicação, um lugar físico como uma sala, um nó *fog* ou um dispositivo IoT. Desta forma, as informações de estado são produzidas pelo processamento de dados brutos e incluem, mas vão muito além da localização de um dispositivo. Por exemplo, o contexto pode englobar o estado da rede (conexão, banda, congestionamento), a situação de um dispositivo IoT (energia residual, modo de operação, etc), o estado de um nó *fog* (carga de trabalho, uso de CPU e memória, etc), bem como informações que determinam o perfil da aplicação e do usuário. A plataforma necessita coletar, processar e armazenar essas informações com o objetivo de realizar ações ou reagir a estímulos do meio do qual tem interesse.

Uma característica de *fog computing* é que seus nós computacionais não necessariamente precisam ser fixos, ou seja, eles podem ser móveis [28]. Considerando um cenário de cidade inteligente onde nós *fog* são acoplados a veículos que se locomovem pela cidade, a plataforma deve assegurar a devida comunicação com esses nós e gerenciar sua disponibilidade, mantendo a continuidade dos serviços ofertados mesmo quando eles estiverem em trânsito por diferentes redes de acesso [29]. A mobilidade envolve o processo de *handover* (também conhecido como *handoff*). Um nó *fog* móvel realiza um *handover* quando ele muda de canal de conexão para outro canal mantendo-se na mesma rede. Para **suportar a mobilidade**, o mecanismo de *handover* precisa cancelar o registro de um nó de um ponto de acesso de origem e registrá-lo em um novo ponto de acesso [29]. Atualmente muitos trabalhos propõem soluções baseadas em LISP (do inglês, *Locator/ID Separation Protocol*), Mobile IPv6[2] e Mobile IPv4[3] para lidar com a mudança dos endereços IP à medida que os nós *fog* se movimentam, desta forma assegurando o suporte a mobilidade [29, 30]. Estas soluções desacoplam a identificação do dispositivo (a qual rede ele pertence no momento) da identificação da sua localização (endereçamento de localização na rede) usando um sistema de indexação distribuído, mantendo informações de um dispositivo independentemente da sua localização geográfica. A mobilidade do nó *fog* não deve ser confundido com o processo de *offloading* de processamento e de dados que pode ocorrer em *fog computing*, conforme discute [31]. *Offloading* pode envolver a migração de processamento computacional atrelado a uma tarefa quando um nó *fog* não é capaz de realizá-la, seja por falta de recursos ou pelo tempo necessário para execução da tarefa. Também, *offloading* envolve a migração de dados de um nó *fog* para outro nó *fog* ou para nuvem, que pode ser motivado por exemplo, por questões de segurança, disponibilidade dos dados ou restrições de armazenamento no nó *fog*, etc.

No contexto de *fog computing*, algumas aplicações críticas como, por exemplo,

[2]Mobile IPv6 - https://tools.ietf.org/html/rfc6275

[3]Mobile IPv4 - https://tools.ietf.org/html/rfc3344

telemedicina e condução assistida (do inglês *assisted driving*), requerem **disponibilidade** contínua, isto é, as plataformas precisam assegurar a oferta de seus serviços mesmo sob condições adversas de operação, evitando assim a degradação dos serviços ao longo do tempo. Técnicas como redundância e duplicação de nós *fog*, isolamento de falhas e aprendizado de máquina são usadas para ajudar a melhorar o MTTR (do inglês *Mean Time To Repair*), que é a média do tempo necessário para a plataforma *fog* se recuperar de uma falha.

Outro proeminente requisito mencionado na literatura é a **diminuição do consumo de energia elétrica**. Este requisito está alinhado ao conceito de sustentabilidade, que preza pela preservação do planeta e ao atendimento das necessidades humanas [32]. Posto isto, a distribuição do processamento em dispositivos de recursos limitados e, consequentemente com baixo consumo energético, torna toda a solução com menor custo energético quando comparado à computação em nuvem, conforme aponta [33]. Alguns trabalhos nesta linha visam otimizar a comunicação de controle e dados entre os nós *fog*, usando protocolos de baixo *overhead* a fim de aumentar a vazão de dados (do inglês *throughput*) de todo o sistema e, por conseguinte, minimizar o consumo energético. Outra técnica promissora é o uso de MIF (do inglês *Multilevel Information Fusion*) para reduzir o volume de dados que precisam fluir entre as camadas *fog* até atingir a nuvem [34].

**Segurança e Privacidade** é a capacidade da plataforma de proteger as informações das aplicações contra acessos não autorizados, além de manter a integridade e a privacidade. Segundo [35], devido às características de heterogeneidade, distribuição e mobilidade, o ambiente *fog computing* é mais vulnerável e menos seguro que a computação em nuvem. Ademais, as medidas existentes de segurança e privacidade de computação em nuvem não podem ser diretamente aplicadas em *fog computing*. Logo, muitos estudos enfocam na criptografia e autenticação para melhorar a segurança da rede a fim de proteger o ambiente *fog* contra ciberataques. Além disso, Redes Definidas por Software (do inglês *Software-defined Networking* - SDN) e Blockchain têm sido apresentadas como soluções de segurança promissoras para o ambiente *fog* [36].

Por fim, uma plataforma *fog computing* deve assimilar uma carga crescente de requisições e dispositivos IoT, mantendo seu funcionamento correto mesmo com alto volume de utilização, i.e., a plataforma precisa ser escalável. Atender ao requisito de escalabilidade é primordial, pois cada vez mais surgem dispositivos IoT conectados que geram uma grande quantidade de dados que requerem recursos para processamento e armazenamento. As soluções mais simples para endereçar **escalabilidade** são incorporar novos nós computacionais ou escalar internamente os nós com a adição de *hardware*, ou *software*. Porém, nem sempre essas soluções simplistas são possíveis de serem realizadas e, portanto, a plataforma deve empregar outras soluções como, por exemplo, o provisionamento de recursos com vistas a otimização da carga de trabalho dos nós disponíveis.

## 1.3. Arquiteturas de Referência

A primeira arquitetura para *fog computing* foi proposta por Bonomi et. al. em 2012 [17]. Na época, os autores vislumbraram *fog computing* como a extensão das capacidades da nuvem para a borda da rede e propuseram uma arquitetura formada por três camadas: i) a camada inferior formada pelos dispositivos IoT que são capazes de sensoriar e atuar no

ambiente em questão; ii) a camada intermediária, que é a camada *fog*, responsável pelo processamento dos dados localmente e quando da indisponibilidade de processá-los, por encaminhá-los para a nuvem; por fim, iii) a camada superior, formada pelos centros de dados na nuvem.

A arquitetura inicial para *fog computing* proposta por [17] impulsionou o surgimento de várias outras propostas nos últimos anos, algumas mais concretas e outras puramente teóricas [37, 38, 39, 6, 40, 19, 20, 41, 42, 43, 44]. Entretanto, conforme discutem [45, 23], ainda não existe uma arquitetura padronizada universalmente reconhecida, pois o que se percebe é que as arquiteturas diferem não somente quanto ao estilo arquitetural adotado, mas principalmente na forma da organização e tipos de dispositivos usados como nós *fog*, além dos objetivos gerais da arquitetura, métricas usadas para o provisionamento de recursos e serviços gerenciados pela plataforma, além do tipo de rede a qual é aplicável.

A diversidade de propostas de arquitetura para *fog computing* revelam a falta de consenso entre pesquisadores, arquitetos de *software* e profissionais de IoT sobre uma arquitetura unificada para *fog computing*, conforme argumentam os autores em [45]. Desta forma, é importante fazer um balanço das arquiteturas a partir de perspectivas de padronização para interoperabilidade entre sistemas *fog computing*, considerando a experiência na área dos principais fabricantes de *hardware* e *software*. Neste sentido, uma Arquitetura de Referência (AR) é o principal artefato para prover tal interoperabilidade, na medida que uma AR específica de maneira unificada e não ambígua, regras de negócio, estilos/padrões arquiteturais, decisões arquiteturais, boas práticas de desenvolvimento e elementos de *hardware* e/ou *software* necessários para a construção de arquiteturas concretas, que dizem respeito aos sistemas propriamente ditos no domínio em questão [46].

Considerando o importante papel desempenhado por uma AR para facilitar o desenvolvimento de sistemas, promovendo a redução de tempo e custo, além de padronizar as arquiteturas de sistemas em um domínio, esta seção apresentará duas ARs para *fog computing*, a saber: i) a AR proposta em [39] e tida como a primeira arquitetura de referência para *fog computing*; e ii) a AR OpenFog, criada pelo consórcio OpenFog e que tornou-se a normativa IEEE 1934-2018 em agosto de 2018.

### 1.3.1. Arquitetura de Referência (Dastjerdi e Buyya, 2016)

A arquitetura de referência proposta por [39] acrescentou mais duas camadas (*IoT applications* e *Software-defined resource management*) no topo da arquitetura de 3 camadas inicialmente proposta por [17], conforme ilustrado na Figura 1.1. Apesar da simplicidade desta AR, ela tem a sua relevância, pois foi uma das primeiras iniciativas de padronização de arquitetura para *fog computing* que estabeleceu a clara separação entre as camadas *fog*, a camada nuvem, além da camada que compreende os dispositivos IoT da borda da rede. Outra contribuição para a área foi o estabelecimento de um vocabulário adequado.

Conforme a Figura 1.1, a camada inferior é formada pelos dispositivos IoT, *gateways*, sensores e atuadores que são capazes de sensoriar e atuar no ambiente em questão. Estes dispositivos são os principais produtores de dados e utilizam os serviços da camada superior, chamada de *Network*, para comunicação entre eles e também com a nuvem. Nesta AR a camada *Network* se assemelha à camada *fog* da arquitetura proposta

por [17], pois ela é responsável por manter a conectividade dos dispositivos da camada subjacente, realizar o processamento dos dados localmente e quando da indisponibilidade de processá-los, encaminhá-los para a camada superior. A camada *Cloud Services and Resources* é responsável pela gerência dos serviços e recursos disponíveis na nuvem, que serão eventualmente usados para processar os dados IoT que alcançarem esta camada. A camada *Software-Defined Resource Management* gerencia toda a infraestrutura e tenta assegurar a qualidade dos serviços que serão ofertados para as aplicações, endereçando questões relacionadas com o monitoramento, provisionamento, segurança e gestão dos dados IoT. Por fim, a camada IoT *Applications* oferece um conjunto de facilidades - não descritas pelos autores da AR - para os desenvolvedores criarem as aplicações IoT que serão executadas na *fog*.

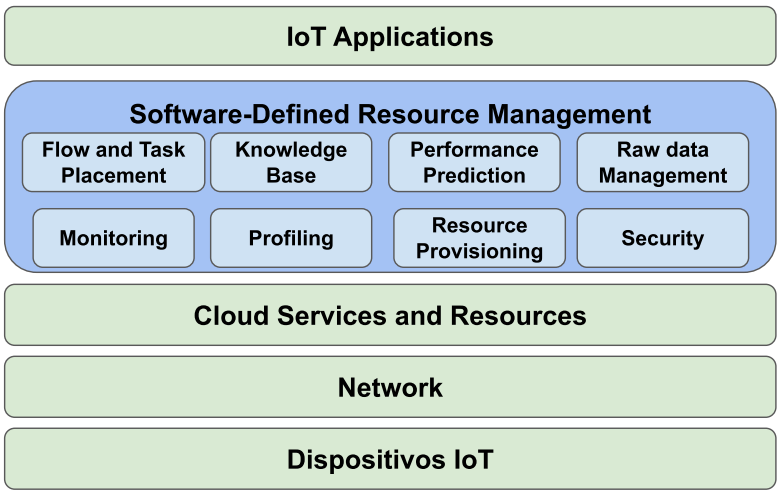



**Figura 1.1. Arquitetura de Referência em cinco camadas proposta por [39]) (adaptado de [39])**

Segundo os autores, dois principais objetivos foram considerados no projeto da AR: i) diminuir o custo financeiro do uso de recursos da nuvem; e ii) propiciar a execução de aplicações sensíveis a latência. Assim sendo, a camada *Software-Defined Resource Management* dispõe de vários componentes que trabalham cooperativamente para ofertar serviços de gerência, controle e execução das aplicações com o intuito de usar de forma apropriada as camadas *fog* (representada como *Network*) e nuvem para atingir os objetivos almejados. Os seguintes componentes estão presentes nesta camada:

- ***Monitoring*** - este componente é responsável por monitorar a execução das aplicações e a utilização dos recursos computacionais da *fog* e da nuvem. Ele fornece informações de telemetria como, por exemplo, a carga de trabalho, quantidade de memória e CPU em uso por cada nó *fog*, bem como o estado de cada aplicação. Estas informações são primordiais para todo o funcionamento do sistema, uma vez que as tarefas que compõem uma aplicação precisam ser distribuídas adequadamente entre os recursos da *fog* e nuvem de maneira a atender aos requisitos da aplicação e também não saturar as capacidades de cada camada.
- ***Flow and task placement*** - este componente é responsável por identificar os locais

apropriados para executar as tarefas das aplicações e lidar com seus respectivos fluxos de dados. Ele faz uso dos serviços providos pelo componente *Monitoring* para obter informações de telemetria dos recursos disponíveis na *fog* e nuvem. Este componente também determina ao componente *Resource Provisioning* que faça o provisionamento dos recursos necessários para atender as demandas. Não existem informações a respeito das métricas e estratégias usadas para posicionar as tarefas nos nós.

- ***Knowledge Base*** - as informações históricas a respeito do uso de recursos e demandas das aplicações são capturadas ao longo do tempo e organizadas em uma base de conhecimento para servir na tomada de decisões de outros componentes. Por exemplo, o componente *Flow and task placement* pode tomar suas decisões sobre a distribuição de tarefas considerando o comportamento histórico das aplicações.

- ***Performance Prediction*** - este componente faz uso da base de conhecimento mantida pelo componente *Knowledge Base* para realizar estimativas sobre o desempenho dos recursos disponíveis na camada de nuvem. Esta informação é útil para o componente *Resource Provisioning* na tomada de decisão sobre o provisionamento de recursos quando existirem muitas tarefas em execução e os critérios de QoS não forem satisfatórios, por exemplo, quando o desempenho (em termos de latência ou banda) exigido por uma aplicação não é satisfeito.

- ***Raw Data Management*** - este componente gerencia o acesso às fontes de dados, abstraindo o acesso a eles por outros componentes da arquitetura. As abstrações fornecidas podem ser simples como, por exemplo, consultas SQL e APIs REST (do inglês *Representational State Transfer*), ou exigirem acessos mais elaborados que envolvam o uso de técnicas como MapReduce.

- ***Security*** - este componente fornece autenticação, autorização e criptografia, conforme requerido pelos serviços da plataforma e aplicações.

- ***Profiling*** - este componente cria perfis que caracterizam as aplicações e os recursos disponíveis a partir da análise das informações obtidas dos componentes Knowledge Base e Monitoring. Os autores não forneceram detalhes destes perfis.

- ***Resource Provisioning*** - este componente realiza o provisionamento de recursos das camadas para a execução das aplicações. O ambiente altamente dinâmico em que plataformas de *fog computing* estão inseridas requer que as decisões de alocação sejam dinâmicas, pois os requisitos das aplicações, bem como o número de aplicações em execução, mudam ao longo do tempo. Portanto, as decisões de quantos e quais recursos alocar devem ser tomadas levando em consideração várias informações que são fornecidas por outros componentes (*Profiling*, *Performance Prediction* e *Monitoring*) além dos requisitos das aplicações. De posse destas informações, o componente pode tomar a decisão, por exemplo, de alocar uma aplicação para ser executada parcialmente na nuvem, pois os dados históricos mostram que tal aplicação possui necessidade de escalabilidade. Por outro lado, uma aplicação talvez não possa ser executada na borda da rede pois os recursos ali presentes estão saturados.

### 1.3.2. Arquitetura de Referência OpenFog

A Arquitetura de Referência OpenFog foi criada pelo consórcio de mesmo nome, com o objetivo de criar uma linguagem padronizada e estabelecer um arcabouço (*framework*) universal para a distribuição dos recursos e serviços, de forma a estimular a interoperabilidade entre aplicações de rede complexas que fazem uso intensivo de dados e que são pertencentes a diversos domínios como IoT, 5G, etc. O consórcio OpenFog foi fundado em novembro de 2015 pelas empresas ARM, Microsoft, Intel, Cisco, Dell e a Universidade de Princeton. Desde sua origem, os grupos de trabalhos do consórcio atuaram de forma conjunta com outros grupos de outros consórcios IoT e tecnologias correlatas como, ETSI-MEC[4] (*Mobile Edge Computing*), OPC-UA[5](*OPC Foundation Unified Architecture*), OCF[6] (*Open Connectivity Foundation*) e OPNFV[7] (*Open Platform for NFV*), com a finalidade de evitar duplicação de esforços e padronizar termos comuns.

Em agosto de 2018 o IEEE (*Institute of Electrical and Electronics Engineers*) adotou a OpenFog como AR para *fog computing* sob o padrão IEEE 1934-2018. Algum tempo depois, em janeiro de 2019, os consórcios OpenFog e IIC[8] (*Industrial Internet Consortium*) anunciaram a sua fusão e a incorporação dos grupos de trabalho nos projetos da IIC. Esta decisão foi tomada pois os dois consórcios trabalhavam com objetivos comuns e, dessa forma, passariam a poder impulsionar o desenvolvimento e a implantação de aplicações *fog computing* para a indústria 4.0.

A AR OpenFog é estruturada em oito pilares e cada pilar representa um princípio ou atributo necessário para a distribuição de computação, armazenamento, controle e funções de rede próximo a origem dos dados. Os pilares são os seguintes:

- **Pilar Segurança**- visa garantir a segurança fim-a-fim de aplicações que serão executadas. Para criar uma ambiente seguro para execução das aplicações, é necessário adotar estratégias de segurança nos dispositivos usados como nós *fog* e na rede que interliga os nós.
- **Pilar Escalabilidade** - visa permitir a adaptação a cargas de trabalho crescentes e diversas estratégias podem ser usadas para assegurar a escalabilidade. Por exemplo, o aumento das demandas de desempenho exigida pelas aplicações requerem o uso de protocolos de comunicação com baixo *overhead*. Além disso, a rede *fog* pode variar seu tamanho à medida que mais aplicações, usuários e dispositivos IoT são adicionados ou removidos. Também, as capacidades podem ser incorporadas aos nós *fog* pela adição de *hardware* como, por exemplo, processadores, dispositivos de armazenamento, aceleradores gráficos e interfaces de rede. A capacidade pode também ser aumentada via *software*, com a adição de serviços e componentes de *software*.
- **Pilar Abertura** - tem por objetivo assegurar a abertura das tecnologias usadas para evitar soluções proprietárias ou de um único fornecedor, o que pode ter um impacto

[4]ETSI-MEC - https://www.etsi.org/technologies/multi-access-edge-computing
[5]OPC-UA - https://opcfoundation.org/
[6]OCF - https://openconnectivity.org/
[7]OPNFV - https://www.opnfv.org/
[8]IIC - https://www.iiconsortium.org/

negativo no custo, qualidade e inovação que a plataforma pode ofertar. A utilização de padrões abertos é essencial para a criação e manutenção de um ecossistema de *fog computing*, onde plataformas e aplicações possam coexistir e compartilhar dados e recursos. Este pilar possibilita por exemplo, que nós *fog* possam existir em qualquer lugar, serem descobertos e dinamicamente criados. Ademais, a comunicação e representação de dados baseada em padrões abertos fortalece a interoperabilidade e habilita a portabilidade de serviços e aplicações na rede *fog*.

- **Pilar Autonomia** - possibilita os nós *fog* fornecerem suas funcionalidades mesmo que outros nós falhem. Diferentemente da computação em nuvem onde as decisões são centralizadas na nuvem, a natureza hierárquica de *fog computing* possibilita que as decisões de implantação de aplicações sejam realizadas em todos os níveis.

- **Pilar Programabilidade** - tem como objetivo assegurar a capacidade do nó ser flexível e mudar a sua configuração e forma de operar para resolver um determinado problema. A reconfiguração de um nó *fog* possibilita a criação de um ambiente dinâmico para execução de diversos tipos de aplicações. Obviamente os nós precisam oferecer às interfaces necessárias para sua reprogramação.

- **Pilar RAS** - o acrônimo RAS (do inglês *Reliability, Availability, and Serviceability*) significa confiabilidade, disponibilidade e operacionalidade, três importantes requisitos não funcionais. A *confiabilidade* diz respeito a entrega da funcionalidade esperada em condições normais e adversas de operação. *Disponibilidade* é definida tradicionalmente como a probabilidade de um sistema ou elemento do sistema estar em um estado funcional no momento em que o usuário precisar. No contexto da AR, o requisitos de disponibilidade assegura o gerenciamento e orquestração contínuas dos nós *fog* para salvaguardar a disponibilidade e integridade dos dados e computação realizadas. Por fim, a *operacionalidade* diz respeito a assegurar o funcionamento correto das funcionalidades ofertadas pela plataforma.

- **Pilar Agilidade** - visa a transformação dos grandes volumes de dados IoT em formatos gerenciáveis e que tenham valor agregado. A agilidade também lida com a natureza dinâmica de *fog computing* e com a necessidade de responder rapidamente às mudanças. Por exemplo, muitos dados IoT podem não ter um contexto definido e consequentemente valor para as aplicações se forem utilizados isoladamente. Porém, após serem coletados, agregados e analisados, tais dados podem apresentar um contexto e representar informações importantes para regra de negócio da aplicação. Se as atividades de coleta, agregação e análise forem realizadas na *fog*, a criação de contexto será mais próximo a geração de dados e decisões operacionais podem ser tomadas mais rapidamente.

- **Pilar Hierarquia** - este pilar assegura diferentes hierarquias que a rede *fog* pode ter para atender as necessidades das aplicações. Por exemplo, uma aplicação que gerencia o controle de acesso aos laboratórios em uma universidade será completamente instalada nos nós *fog* locais, que estão situados no prédio, e possivelmente não fará uso da camada nuvem. Por outro lado, uma aplicação de grande porte para cidades inteligentes como, por exemplo, o controle de tráfego, pode ser instalada em diversos níveis da rede *fog* e distribuída geograficamente na cidade.

A AR OpenFog segue a essência da arquitetura proposta por [17], ao dividir a arquitetura em três camadas, sendo que a camada intermediária é subdividida em quatro outras camadas. Esta subdivisão possibilita que implementações concretas da arquitetura possam residir em diferentes camadas da topologia de uma rede e assegurar os benefícios oferecidos por computação em nuvem como, por exemplo, virtualização, orquestração, eficiência e capacidade de gerenciamento.

#### 1.3.2.1. Descrição da Arquitetura de Referência OpenFog

A descrição da AR OpenFog segue o *guideline* proposto pelo padrão ISO/IEC/IEEE 42010:2011 e utiliza os seguintes termos do vocabulário especificado por este padrão:

- **Ponto de Vista** (do inglês *Viewpoint*) - é uma maneira ou forma de se olhar um sistema. O ponto de vista contém um conjunto de interesses (do inglês *concerns*) e os modelos e técnicas usadas para descrever a arquitetura que atenda estes interesses.
- **Visão** (do inglês *View*) - uma visão é uma representação de um ou mais aspectos estruturais da arquitetura e fornece descrições variadas que mostram a arquitetura sob diferentes ângulos. As visões representam os conceitos de interesse de um conjunto de *stakeholders* e pode ser entendida como uma realização de um ponto de vista que permite reduzir a quantidade de informação para um *stakeholder* em um dado momento.
- **Perspectiva** (do inglês *Perspective*) - representam as questões ou capacidades que permeiam toda a arquitetura, i.e., afetam todas as camadas e componentes da arquitetura. As perspectivas definem decisões arquiteturais relacionadas a atributos de qualidade ou requisitos não funcionais.

A descrição da AR OpenFog, ilustrada na Figura 1.2, é a representação composta pelas diferentes visões que atendem aos anseios dos *stakeholders*, além das diferentes perspectivas.

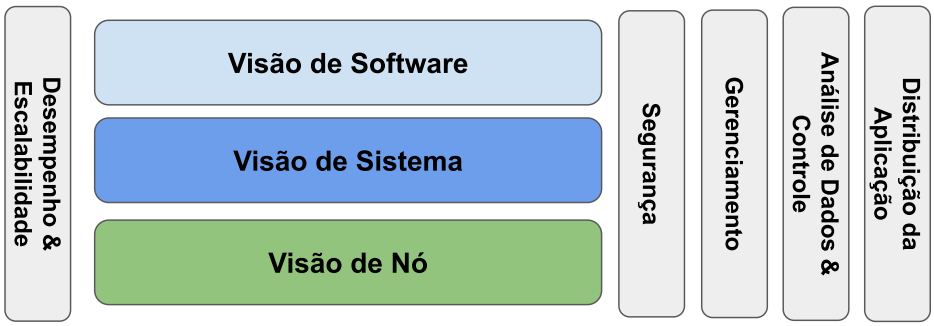



**Figura 1.2. Descrição da AR OpenFog (adaptado de IEEE 1934-2018)**

Na Figura 1.2 as **perspectivas** são identificadas como barras verticais. São elas: i) Desempenho & Escalabilidade; ii) Segurança; iii) Gerenciamento; iv) Análise de Dados

& Controle; e v) Distribuição da Aplicação. Ainda na Figura 1.2, a AR apresenta três diferentes visões, a saber: i) visão de *software*; ii) visão de sistema; e iii) visão de nó. Cada visão é formada por camadas que serão descritas no decorrer deste texto.

Dois pontos de vista são apresentados pela AR: i) **ponto de vista funcional** que demonstra como aplicar os elementos da arquitetura e as diferentes visões para atender as necessidades dos *stakeholders* em um dado cenário; e ii) **ponto de vista de implantação** que demonstra como sistemas *fog* são implantados/instalados para lidar com um determinado cenário. O documento OpenFog apresenta sucintamente no ponto de vista funcional um cenário de vigilância em aeroportos com o uso de câmeras de vídeo e devido a simplicidade deste ponto de vista, não a discutiremos neste texto. As próximas subseções apresentam o ponto de vista de implantação, as diferentes perspectivas e visões da AR.

#### 1.3.2.2. Ponto de Vista de Implantação

A implantação diz respeito à organização lógica hierárquica dos nós *fog* em um determinado momento. Uma característica da *fog computing* é a sua capacidade de mudar a organização lógica para cada cenário diferente, adequando-se às necessidades das aplicações. As implantações podem ser grandes ou pequenas devido aos requisitos do cenário que será endereçado. Vários fatores contribuem para o dimensionamento da implantação como, por exemplo, quantidade e tipos de processamento exigidos por cada camada, quantidade de sensores/atuadores, capacidades de cada nó *fog*, latência na comunicação entre os nós e também entre os sensores e atuadores, bem como a existência de nós redundantes para atender ao requisitos de confiabilidade. O tipo mais comum de organização dos nós *fog* é em camadas (N-camadas), conforme ilustrado na Figura 1.3.

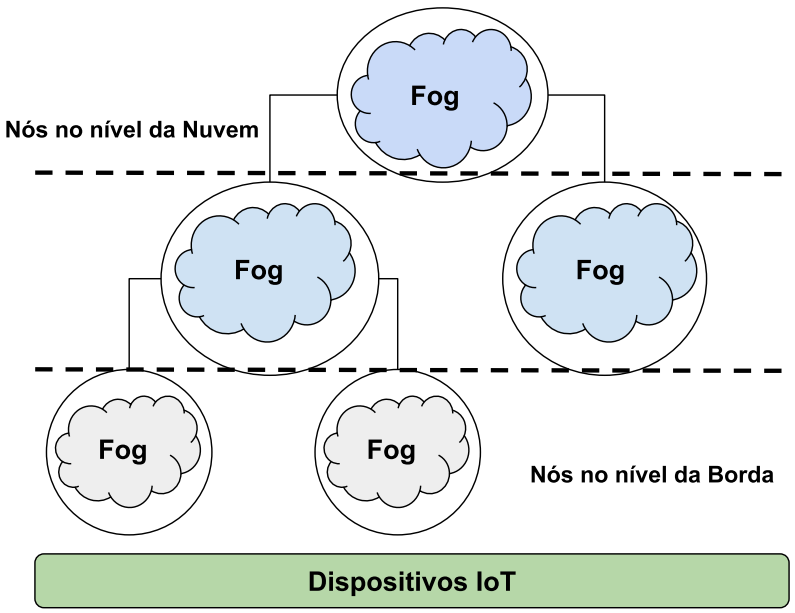



**Figura 1.3. Modelo de implantação em N-camadas (adaptado de IEEE 1934-2018)**

Neste tipo de organização os nós que estão localizados na camada mais próximas a

borda da rede (do inglês *edge network*), tipicamente possuem menor poder computacional e são responsáveis pela aquisição, coleta e normalização dos dados dos dispositivos IoT, e também o controle dos sensores e atuadores acoplados a estes dispositivos. Os nós que estão na camada sobrejacente, realizam a filtragem, compressão e transformação dos dados advindos da camada subjacente. Também, nesta camada podem ocorrer processos de análise de dados e inferências (do inglês *Data Analytics*) para tomada de decisões mais rápidas, a fim de endereçar requisitos de sistemas de tempo real ou próximo a tempo real. Entretanto, as análises são simples pois neste nível os recursos computacionais e as capacidades de inferência normalmente são limitadas. Por fim, os nós *fog* que estão na camada mais alta ou próximas a nuvem são, normalmente, responsáveis pela agregação de dados de forma a gerar conhecimento.

De maneira geral, no modelo N-camadas cada camada filtra e extrai dados significativos para criar mais inteligência. Desta forma, as camadas inferiores da hierarquia tratam do aquisição dos dados IoT e alguns processamentos simples nestes dados, enquanto que as camadas superiores têm interesse na criação de inteligência. Entretanto, em alguns modelos de implantação como, por exemplo, para análise de vídeo de câmeras de segurança, a análise - que é um processamento de inteligência - deverá ser realizada na borda da rede, pois a largura da banda de rede pode não ser suficiente para enviar todo *stream* de vídeo para as camadas superiores. Desta forma, realizando os processos de análise na borda da rede a quantidade de dados que necessitam ser enviados para as camadas superiores será menor.

#### 1.3.2.3. Perspectivas

A AR OpenFog define cinco perspectivas e cada perspectiva representa as capacidades ou requisitos de qualidade que permeiam todas as camadas da arquitetura *fog computing*, a saber: i) Desempenho & Escalabilidade; ii) Segurança; iii) Gerenciamento; iv) Dados, Análise & Controle; e v) Distribuição da Aplicação. As perspectivas também envolvem decisões de projeto que devem ser tomadas para atender as demandas dos *stakeholders*.

A **Perspectiva de Desempenho & Escalabilidade** possibilita a execução de processos de inteligência e análise - que antes tipicamente eram executados na nuvem - na borda da rede ou perto dela e, consequentemente próximo aos produtores de dados, o desempenho de todo o sistema *fog computing* será melhorado. Outra característica de *fog computing* é possibilitar às aplicações uma rápida adequação às mudanças de tráfego de dados, por exemplo, o surgimento de carga crescente de dados IoT advindos de vários sensores que controlam o tráfego de uma cidade no horário de pico. Desta forma, o sistema *fog computing* pode detectar esta alteração mais rapidamente e escalar todo o sistema ou parte dele. Porém, ao escalar uma parte do sistema *fog computing*, deve ser garantido que não haverá a interrupção ou diminuição na oferta de serviços da plataforma e na execução de outras aplicações.

A **Perspectiva de Segurança** preocupa-se com a segurança fim-a-fim que é um elemento crítico para o sucesso de uma plataforma *fog computing* e envolve todos os dispositivos que estão entre a nuvem e os dispositivos finais. Os dispositivos IoT e nós *fog* necessitam ser seguros, bem como a rede formada por eles também necessita ser segura e

confiável. Deste modo, nos diferentes modelos de implantação (ver seção 1.3.2.2) se uma camada não for segura e as demais camadas forem seguras (ou vice versa), toda a solução não será segura. Sistemas confiáveis dependem da utilização de elementos de confiança que são responsável pela manutenção das políticas de segurança para os seus dispositivos e a realização da inspeção do comportamento dos dispositivos para determinar se o sistema e seus componentes estão operando de forma confiável. As políticas de segurança especificam quem ou o que pode acessar os componentes e dispositivos e sob quais circunstâncias pode ocorrer o acesso. Além do mais, as diferentes implantações de *fog computing* requerem diferentes mecanismos de segurança que são dependentes das ameaças e do valor do ativo que precisa ser protegido em um determinado nó *fog*. As ameaças compreendem os diferentes tipos de ataques que podem prejudicar um ativo ou ocasionar uma falha de segurança na plataforma [47]. São exemplos de ameaças, ataques diretos ao *hardware*, ataques aos componentes de *software* e de rede, e ataques internos que partem de dentro da rede *fog*. As implicações desses ataques, normalmente, dizem respeito à violação da confidencialidade, integridade, autenticidade, disponibilidade e privacidade dos dados. Por sua vez, os ativos podem envolver dados financeiros, informações sensíveis e pessoais, propriedade intelectual, informações da própria infraestrutura da plataforma, etc.

Na **Perspectiva de Gerenciamento** os nós utilizados em uma implantação *fog computing* podem ser instalados em ambientes remotos, fixos e não-fixos, e em ambientes com condições extremas como, por exemplo, no domínio de agronegócio os nós *fog* podem ser instalados em equipamentos agrícolas e estarão sujeitos às intempéries do clima. Ademais, os nós *fog* podem precisar de atualização de *software* e/ou *firmware*. Desta forma, configurar e gerenciar manualmente os nós é praticamente inviável e, portanto toda implantação *fog* precisa fornecer uma entidade de gerenciamento. Muitas plataformas usam *heartbeat* como estratégia para gerenciar a saúde das implantações. Nessa estratégia, os componentes da plataforma precisam enviar periodicamente uma informação de controle atestando sua saúde para a entidade de gerenciamento da plataforma. Ainda na perspectiva de gerenciamento, todos os nós *fog* seguem um ciclo de vida de gerenciamento formado por cinco fases, a saber: i) *Ativação*; ii) *Provisão*; iii) *Operação*; iv) *Recuperação*; e v) *Desativação*. O ciclo de vida deve ser assegurado por agentes de gerenciamento em cada nó. Na fase *Ativação*, são realizadas algumas ações para a correta identificação do nó *fog*, verificação de certificados, calibração de relógios, atestação e autenticidade do nó, etc. A fase *Provisão*, inclui atividades como descoberta, identificação e anúncio de recursos ou capacidades do nó. Na fase de *Operação*, os nós estão em funcionamento normal, mas no momento em que eles começam a funcionar de forma não esperada, então a fase de *Recuperação* se inicia. O processo de recuperação precisa ser autônomo, i.e., o próprio nó precisa realizar operações de recuperação e, eventualmente, outros nós podem ajudá-lo na ação de recuperação. Por fim, na *Desativação*, o nó precisa, muitas vezes, apagar informações atualmente usadas para uma implantação de forma a deixá-lo pronto para utilização em outras implantações.

Na **Perspectiva de Dados, Análise & Controle** a análise (do ingês *analytics*) é o processo de criar conhecimento a partir de dados e informações. O modelo tradicional de análise centrado na nuvem, em muitos casos, não é viável devido ao grande volume de dados que precisa ser plenamente capturado, armazenado e transportado. *Fog com-*

*puting* permite capturar, armazenar e transportar apenas dados relevantes à medida que tem a capacidade de realizar processos de análise no continuum. Quatro tipos de análises podem ser realizadas em diferentes camadas da *fog* de acordo com as necessidades das aplicações: i) a *análise descritiva* é realizada nas camadas mais baixas e é usada normalmente pelas aplicações para compreender o estado atual de suas operações, i.e., a análise do que está acontecendo ou do que aconteceu; ii) a *análise diagnóstica* tem como objetivo apresentar o que motivou determinado evento; iii) a *análise preditiva* utiliza todo o conhecimento da análise anterior juntamente com outros conhecimentos da regra de negócio da aplicação para entender o que irá acontecer; por fim,a iv) *análise prescritiva* é utilizada para melhorar os processos de análise.

A **Perspectiva Distribuição da Aplicação** tenta assegurar um ecossistema *fog computing* formando por múltiplos fornecedores (do inglês *multi-vendor*), onde as aplicações precisam ser espalhadas e ter a capacidade de migrar e operar/executar corretamente em qualquer nível da hierarquia de uma implantação *fog*. Desta forma, as aplicações precisam interoperar com todos os níveis da implantação. Assim, os dados que são coletados ou gerados por um nó *fog* devem ser compartilhados com outros nós na hierarquia. Por exemplo, uma aplicação pode ser executada em diferentes nós *fog* que requerem a comunicação entre eles. Assim, os nós que vão se comunicar podem estar situados na mesma camada na hierarquia *fog*, configurando uma distribuição leste-oeste. Por outro lado os nós podem estar em camadas diferentes, o que configura uma distribuição norte-sul.

#### 1.3.2.4. Visões

Conforme dito anteriormente as visões representam os conceitos de interesse de um conjunto de *stakeholders* e pode ser entendida como uma realização de um ponto de vista que permite reduzir a quantidade de informação para um *stakeholder* em um dado momento. A AR OpenFog apresenta três visões que serão discutidas nas próximas subseções.

##### 1.3.2.4.1. Visão de Nó

A visão de nó é formada pelas duas camadas (*Abstração dos Protocolos* e *Sensores e Atuadores*) mais baixas da representação da AR, exibidas em cor verde, além de alguns aspectos, em azul, conforme ilustrado na Figura 1.4.

A inserção de um nó na rede *fog computing* requer o endereçamento de várias aspectos, a saber: i) Segurança; ii) Gerenciamento; iii) Rede; iv) Aceleradores de *Hardware*; v) Computação; e vi) Armazenamento. Estes aspectos são discutidos em seguida:

- **Segurança** - a segurança de cada nó *fog* é essencial para a segurança de todo o sistema *fog computing*, conforme dito na descrição da Perspectiva de Segurança (seção 1.3.2.3). Em alguns casos um nó *fog* pode servir também como *gateway* para sensores e atuadores legados que não implementam mecanismos de segurança. Os nós *fog* estarão, em muitos casos, em espaços físicos públicos e estão sujeitos a alterações indesejáveis. Desta forma, o nó precisa empregar mecanismos para a detecção e notificação de adulterações de forma a garantir a confiabilidade do nó e, por conseguintes, de todo sistema *fog*.

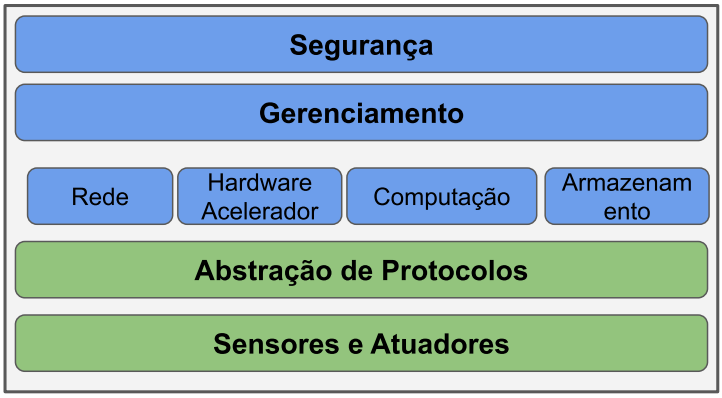



**Figura 1.4. Visão de Nó (adaptado de IEEE 1934-2018)**

- **Gerenciamento** - o gerenciamento do nó é somente realizado se o nó provê uma interface de gerenciamento. Tal interface precisa implementar um protocolo de gerenciamento para possibilitar a agentes de gerenciamento averiguar a saúde e controlar a configuração interna do nó. As informações de saúde podem envolver informações dos elementos internos do nó como as unidades de armazenamento, CPU, memória e o estado dos *hardwares* acelerados, a temperatura, a voltagem, etc. A configuração interna do nó pode incluir, por exemplo, parâmetros de configuração como endereço IP, e configurações dos *hardwares* acelerados.

- **Rede** - todo nó *fog* precisa estar interconectado por uma rede (*network*), que deverá prover a escalabilidade e conectividade requeridas para as diferentes implantações *fog computing*. O modelo de conectividade da rede dependerá da finalidade e localização do nó. Por exemplo, um nó *fog* que está situado em uma fábrica e é usado para capturar e analisar dados relativos às operações da fábrica, provavelmente será conectado as camadas da rede *fog* por uma rede cabeada. Por outro lado, um nó *fog* usado para capturar e analisar informações de localização de veículos em uma cidade será conectado aos sensores nos veículos por uma rede sem fio. Também, os nós *fog* podem ter conexões diretas entre eles usando tecnologias de interconexão de baixa latência, como RDMA (do inglês *Remote Direct Memory Access*). As redes sem fio desempenham um papel muito importante na tecnologia IoT. Existem três principais tipos de redes sem fio que *fog computing* pode fazer uso dependendo do modelo de implantação, a saber: i) WWAN (*Wireless WAN*); ii) WLAN (*Wireless LAN*); e iii) WPAN (*Wireless Personal Area Networks*). Cada tipo de rede possui seus protocolos e padrões. A WWAN é usado para cobrir uma grande área geográfica. São exemplos de WWAN as tecnologias de comunicação celular como, 3G, 4G LTE e 5G, os padrões NB-IoT (*Narrow Band IoT*) e LPWAN (*Low-Power Wide Area Networks*). A rede WLAN, também conhecida como WiFi, são apropriadas para espaços geográficos menores e velocidades mais altas que os demais tipos de rede, sendo que os padrões mais utilizados pertencem a família IEEE 802.11 (a, b, g, n, ac). A WPAN são redes menores que requerem menor consumo energético. São exemplos deste tipo de rede, Bluetooth, comunicação via Infravermelho (IR), ZigBee, e o padrão IEEE 802.15.4 (*Low Rate WPAN*).

- ***Hardwares* aceleradores** - algumas aplicações necessitam de *hardwares* aceleradores como, por exemplo FPGA (do inglês *Field-Programmable Gate Array*) e GPGPU (do inglês *General Purpose Graphics Processing Unit*) para satisfazer requisitos de latência e poder computacional no processamento de grandes volumes de dados. Tais *hardwares* permitem por exemplo, operações paralelas entre matrizes gigantescas reduzindo consideravelmente o tempo computacional.

- **Computação** - o nó *fog* também necessita ter capacidade para computação de propósito geral. Desta forma, o nó *fog*, em muitos casos, necessita ter um conjunto de bibliotecas e APIs padrão para que as aplicações possam ser executadas adequadamente. Por exemplo, muitas plataformas fazem uso de virtualização leve - como Docker - para execução das aplicações que são encapsuladas como contêineres. Desta forma, o nó *fog* precisa fazer o devido controle da execução dos contêineres sobre o uso dos recursos disponíveis como processadores, memória e dispositivos de entrada e saída.

- **Armazenamento** - O armazenamento pode ser de dados de sensores que estão diretamentes associados ao nó *fog*, dados de contexto, *logs* e dados de aplicações que estão residindo/executando atualmente no nó. Muitas tecnologias de armazenamento podem ser usadas pelos nós *fog*, por exemplo, RAM Arrays para armazenamento em memória de forma a diminuir o tempo de acesso aos dados, uso de dispositivos de estado sólido (SSD do inglês *Solid-state Drive*) e discos rígidos que podem eventualmente ter organizações específicas como, por exemplo, serem organizados em uma única unidade lógica para formar um sistema de armazenamento, como propõe RAID (*Redundant Array of Independent Disks*).

A **camada Sensores e Atuadores** compreende os sensores e atuadores que são os elementos de *hardware* ou *software* (pois podem ser virtualizados) fundamentais na IoT. Vários deles, até mesmo milhares, podem estar associados a um único nó *fog*. Existem uma infinidade de diferentes dispositivos IoT, desde dispositivos do tipo *dumb*, que não têm capacidade de processamento significativa, até dispositivos que apresentam as capacidades necessárias para executar funções da *fog*, como por exemplo, os dispositivos Raspberry da família Pi e NVIDIA Jetson Nano. Os dispositivos possuem capacidades de conectividade via cabo via sem fio, e fazem uso de diferentes meios e protocolos de comunicação como, I2C, GPIO, SPI, BTLE, ZigBee, USB e Ethernet. Devido às restrições de conectividade de alguns sensores e atuadores, eles não são capazes de interagir diretamente com um nó *fog*. Desta maneira, a **camada de Abstração de Protocolos** tem como objetivo tornar possível a conexão deles com um nó*fog*com o intuito que os dados dos sensores possam ser adequadamente capturados, bem com os comandos possam ser enviados aos atuadores. Também, esta camada precisa assegurar o acesso às políticas de segurança, conforme descrito na perspectiva segurança.

#### 1.3.2.4.2. Visão de Sistema

A visão de sistema é formada pela junção de uma ou mais *Visões de Nó* e as camadas *Virtualização de Hardware* e *Plataforma de Hardware*, conforme ilustrado na Figura 1.5. Esta visão proporciona uma representação do sistema *fog* e, apesar de que a Figura 1.5

ilustra apenas um nó *fog* na visão, não se descarta a possibilidade de existirem vários nós para criar um sistema *fog*.

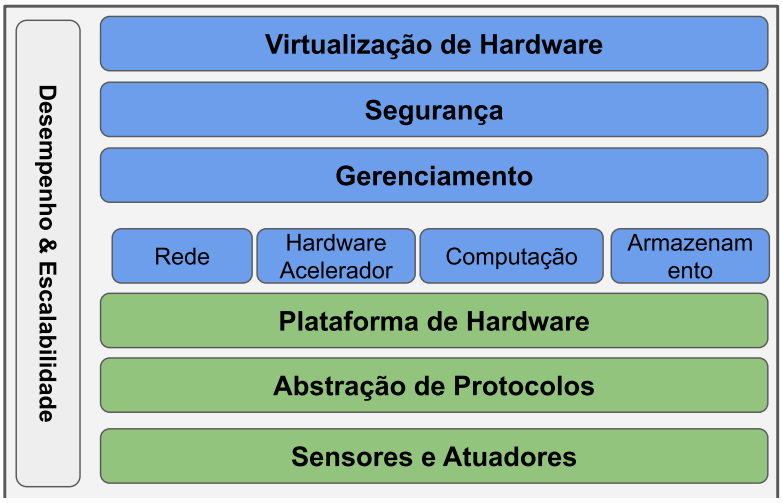



**Figura 1.5. Visão de Sistema (adaptado de IEEE 1934-2018)**

Na visão de sistema, a **camada Plataforma de *Hardware*** representa os dispositivos físicos da plataforma *fog computing*. Tais componentes podem estar inseridos em ambientes com condições severas e, portanto requerem proteção física. Por exemplo, no contexto de cidades inteligentes, nós *fog* podem ser instalados em postes que são usados para iluminação pública de forma que os nós estarão sujeitos às intempéries do clima, poluição, vandalismo, etc. Desta maneira, várias questões precisam ser atendidas para manter uma boa infraestrutura de *hardware*. São exemplos: a conformidade com regulamentos locais e padrões como emissão de calor e ruído; proteção contra fatores ambientais (temperatura, umidade, radiação solar, vibrações, quedas, etc); resistência a ataques físicos, vandalismo ou roubo; adequação de tamanho e peso do *hardware* e consumo de energia elétrica; suporte mecânico para acesso aos componentes internos mais facilmente; e refrigeração dos componentes internos.

A **camada Virtualização de *Hardware*** representa os mecanismos de virtualização, sejam eles baseados em virtualização completa (todo o sistema) ou leve (apenas a aplicação), que são essenciais para *fog computing*, pois permitem, por exemplo, que diferentes aplicações possam coexistir no mesmo nó *fog* e compartilhar o mesmo *hardware* e/ou sistema operacional, sem interferir nas operações (por exemplo entrada e saída) e processos de regra de negócio das outras aplicações.

#### 1.3.2.4.3. Visão de *Software*

A visão de *software* demonstra a execução de um *software* em uma plataforma formada por uma ou mais visões de nó juntamente com outros componentes para endereçar um determinado cenário *fog computing*. A visão de *software* é representada pelas três camadas superiores (**Serviços de Aplicação**, **Suporte de Aplicação**, e **Gerenciamento do Nó & Execução de *Software***) da descrição AR OpenFog. Essas três camadas estão acima da camada **Plataforma de *Hardware***, conforme ilustrado na Figura 1.6.


**Figura 1.6. Visão de *Software* (adaptado de IEEE 1934-2018)**

A **camada Gerenciamento do Nó & Execução de *Software*** é responsável pelas operações e serviços de gerenciamento do nó, mantendo os elementos de *hardware* e *software* configurados adequadamente para o modelo de implantação em uso, além de manter o nó em funcionamento em níveis especificados previamente de disponibilidade, resiliência e desempenho. Outra responsabilidade da camada é fornecer suporte a execução de *software* no nó e facilitar a comunicação entre os nó e os demais, de forma a atender a perspectiva de distribuição da aplicação. Nesta camada, o *software* em execução representa uma aplicação *fog computing* ou parte dela, que tipicamente são executadas em ambientes virtualizados ou em contêineres.

A **camada Suporte a Aplicação** é formada por um conjunto de *software* e sistemas que fornecem serviços de infraestrutura que não são relacionados a nenhum domínio de problema ou aplicação, mas que servem para ajudar e apoiar serviços de aplicação (*software* de aplicação). São exemplos de *software* e sistemas pertencentes a esta camada, motores de execução de máquinas virtuais e contêineres (Docker, VMs, etc), API(do inglês *Application Programming Interface*) e bibliotecas de linguagens de programação, *software* e sistemas para comunicação assíncrona e síncrona (ActiveMQ, RabbitMQ, DDS, etc), sistemas para persistência (MySQL, SQLite, Cassandra, MongoDB, Redis, etc), ferramentas e *frameworks* para análises (Spark, Hadoop, etc), etc.

Por fim, a **camada Serviços de Aplicação** representa o conjunto de *software* que são dependentes das camadas subjacentes e são projetados para endereçar requisitos específicos de domínio e de uma aplicação. Isto é, são as aplicações *fog computing*.

## 1.4. Plataformas open source de *fog computing*

Esta seção discute as principais plataformas de *fog computing* de código aberto (*open source*). Tal discussão será embasada pelos requisitos previamente levantados para plataformas de *fog* (ver seção 1.2) e uma comparação entre as plataformas será apresentada. Para a escolha das plataformas foram utilizados os seguintes critérios: i) quantidade de citações do artigo científico que apresenta a plataforma; ii) a atualidade do repositório de código fonte, i.e., o quão ativo e atual é o projeto da plataforma e se ela ainda está em evolução; e iii) a disponibilidade plena da plataforma, pois algumas plataformas são apresentadas apenas em artigos científicos e não estão disponíveis para uso, enquanto outras não tiveram seus projetos continuados e estão defasadas. De posse destes critérios serão descritas as plataformas FogFlow, Stack4Things, Microsoft Azure IoT Edge e ParaDrop.

### 1.4.1. FogFlow

A plataforma FogFlow [7] faz parte do projeto FIWARE[9] e serve para orquestrar a carga de trabalho de aplicações em nós *fog* considerando a localização geográfica dos produtores de dados e dos nós *fog*, de forma a tratar dos requisitos de otimização de uso da largura de banda e suportar aplicações sensíveis à latência. FIWARE é uma plataforma ampla formada por vários componentes genéricos, interoperáveis e extensíveis, chamados de GEs (do inglês *Generic Enablers*). Os GEs fazem uso do protocolo NGSI (*Next Generation Service Interfaces*) que especifica a forma de acesso aos serviços ofertados pelos componentes da plataforma FIWARE, bem como define um modelo de representação dos dados de contexto. O protocolo NGSI é mantido pela OMA (*Open Mobile Alliance*) e define a interface de comunicação para troca de informações contextuais entre diferentes GEs, i.e., promove a interoperabilidade no ecossistema FIWARE. O modelo de representação de dados de contexto NGSI utiliza o conceito de entidades de contexto, que são representações virtuais de objetos, sejam eles físicos ou não, e suas respectivas propriedades. Por exemplo, uma sala de aula pode ser representada como uma entidade de contexto com as seguintes propriedades: quantidade de pessoas; temperatura; localização geográfica; e luminosidade.

FogFlow fornece um modelo de programação onde a lógica de processamento de cada aplicação é encapsulada em *tarefas* (do inglês *tasks*) e cada *tarefa* possui entradas e saídas representadas como *entidades de contexto*. Desta forma, as aplicações são definidas como um grafo dirigido de tarefas (DAG, do inglês *Directed Acyclic Graph*) que executam a lógica de processamento dos dados. O resultado do processamento de uma *tarefa* é uma entidade de contexto que servirá como entrada para outra *tarefa*. Cada tarefa é implementada como uma imagem Docker e a plataforma se responsabiliza pela orquestração das tarefas considerando a topologia (organização lógica) e características dos nós *fog*. Todos os componentes desta plataforma comunicam-se através de um *broker* de comunicação compatível com o protocolo NGSI e cabe ressaltar que a plataforma espera que todos os dados IoT sigam o formato NGSI. Entretanto, muitos dispositivos IoT são simples e incapazes de publicar e/ou receber dados no formato NGSI devido, principalmente, a sua limitação de processamento. Os nós *fog* utilizados pela plataforma para a distribuição das múltiplas tarefas são chamados de *workers* e cada *worker* atua como um provedor para publicar o resultado de sua computação no *broker*, e também atua como consumidor no *broker* para receber dados de contexto ou novas tarefas para serem executadas.

A arquitetura da plataforma FogFlow, ilustrada na Figura 1.7, é baseada em três camadas. A **camada Gerenciamento de Serviço** (do inglês *Service Management*) está localizada na nuvem e contém os componentes para o gerenciamento de toda a plataforma e para criação de aplicações. Nesta camada, o componente *Repositório de Tarefas* é responsável por armazenar todas as descrições tarefas que compõem as aplicações, sendo que cada tarefa é implementada e encapsulada como uma imagem Docker. O componente *Designer de Tarefas* oferece uma interface gráfica Web para a especificação das aplicações em termos de tarefas e entidades de contextos. Isto é, o *Designer de Tarefas* ajuda o desenvolvedor a criar visualmente o DAG que representa a aplicação. Por fim, o componente *Orquestrador* é responsável por fazer a distribuição das tarefas que com-

[9]FIWARE - https://www.fiware.org/foundation/

põem uma aplicação nos nós *fog* mantidos pela plataforma. A distribuição das tarefas é guiada pela localização geográfica da fonte de dados de interesse da aplicação. Uma vez que a plataforma conhece previamente a localização geográfica de cada nó *fog* e também da fonte de dados, o componente *Orquestrador* tenta associar o nó *fog* mais próximo à fonte de dados para executar as tarefas da aplicação. Além da informação de localização, o *Orquestrador* também utiliza a informação de capacidade de execução simultânea de tarefas de cada nó, de forma a não sobrecarregá-los, bem como sua arquitetura (X86 ou ARM) para verificar a compatibilidade da imagem Docker correspondente a tarefa.

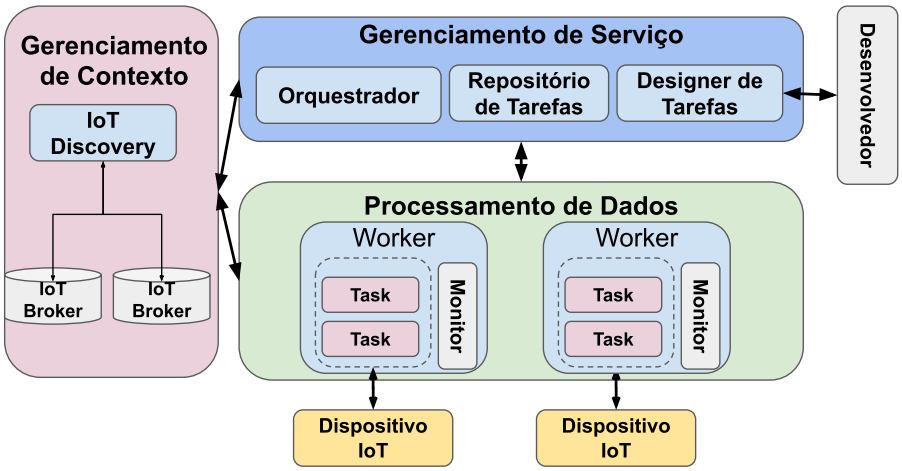



**Figura 1.7. Arquitetura da plataforma FogFlow (adaptado de [7])**

A **camada de Processamento de Dados** (do inglês *Data Processing*) compreende os nós *fog* que são responsáveis pela execução das tarefas demandadas pelo componente *Orquestrador*. Na plataforma FogFlow cada nó *fog* (*Worker*), ao receber a requisição para a execução de uma dada tarefa, solicita a imagem Docker correspondente da tarefa ao componente Repositório de Tarefas. Logo em seguida, após obter a imagem, o *Worker* instancia a imagem e executa o contêiner Docker (tarefa) no ambiente virtualizado. Os nós *fog* podem estar localizados na borda da rede ou também podem ser completamente virtualizados na nuvem. Cada nó possui uma capacidade diferente de execução de tarefas que é levada em consideração no ato da orquestração das tarefas pelo *Orquestrador*. Também, em cada *Worker* existe um componente chamado *Monitor* que é responsável por inspecionar o dispositivo e recuperar dados de telemetria (quantidade de tarefas em execução, consumo de memória, processador e armazenamento, etc) que são submetidos periodicamente ao *Orquestrador*.

Por fim, a **camada Gerenciamento de Contexto** (do inglês *Context Management*) tem como objetivo gerenciar todas as entidades de contexto e torná-las passíveis de descoberta e acessíveis via consulta e/ou subscrições. O FogFlow mantém o componente centralizado *IoT Discovery* que provê uma visão global das entidades de contexto disponíveis para todos os outros componentes da plataforma e também para as tarefas em execução. O *IoT Discovery* não armazena os valores das entidades de contexto, ele serve apenas como um elemento indexador que sabe onde recuperar tais informações. As entidades de contexto são gerenciadas pelas várias instâncias do componente *IoT Broker*.

Idealmente, devem existir várias instâncias de *IoT Broker* distribuídas geograficamente, de tal forma que um dispositivo IoT possa publicar seus dados no *IoT Broker* mais próximo a ele. Por sua vez, as tarefas também precisam se subscrever em um *IoT Broker* para receber as entidades de contexto de seu interesse, assim como publicar o resultado do seu processamento (uma entidade de contexto). Neste sentido, é comum ter um *IoT Broker* para cada nó *fog* disponível. Cada instância do componente *IoT Broker* armazenam em memória somente o último valor de cada entidade de contexto. Além do mais, devido a dinâmica de orquestração da plataforma, muito provavelmente a tarefa consumidora de dados (subscribe) estará associada no mesmo *IoT Broker* que o dispositivo produtor de dados (*publisher*).

Considerando os requisitos previamente levantados para plataformas *fog computing*, identificamos que a plataforma não apresenta elementos para endereçar os requisitos de suporte a mobilidade e eficiência energética. Enquanto, o requisito de segurança e privacidade é atendido parcialmente, pois a plataforma pode fazer uso do protocolo de comunicação segura HTTPs (*Hyper Text Transfer Protocol Secure*) na comunicação entre os componentes da nuvem e os nós *fog*, mas não especifica a comunicação segura entre os dispositivos IoT e o *broker*. Por sua vez, o requisito de escalabilidade também é atendido parcialmente, uma vez que a plataforma não oferece mecanismos para escalar automaticamente como, por exemplo, manter um conjunto de nós *fog* em estado de espera para serem utilizados apenas quando a demanda crescer ou então implementar estratégias de duplicação de tarefas em nós *fog* diferentes para assegurar também a disponibilidade. Por fim, o protocolo NGSI é baseado em HTTP e, conforme [48], este protocolo não é adequado para aplicações ultra sensíveis a latência devido ao *overhead* exigido pelo protocolo TCP (*Transmission Control Protocol*) no estabelecimento e manutenção da conexão.

### 1.4.2. Stack4Things

Stack4Things [49, 50] é uma plataforma baseada no OpenStack[10] que tem como objetivo fornecer recursos de sensoriamento e atuação como serviços, implementando a ideia de SAaaS (do inglês *Sensing-and-Actuation-as-a-Service*) [51]. A plataforma realiza o provisionamento sob demanda dos recursos e provê uma representação uniforme dos dispositivos IoT distribuídos geograficamente em uma cidade, abstraindo e gerenciando-os em um ecossistema unificado de objetos a serem configurados de acordo com os requisitos das aplicações. A plataforma mapeia os requisitos das aplicações em níveis mais baixos correspondentes as capacidades dos nós *fog*.

Conforme ilustrado na Figura1.8, a arquitetura da plataforma é baseada em duas camadas. A **camada IaaS** (do inglês *Infrastructure-as-a-Service*) está localizada na nuvem e é responsável por manter os componentes *Virtual Boards*, que são representações virtuais dos nós *fog*. A plataforma faz uso de uma abordagem SDN (do inglês *Software Defined Network*) para associar a virtualização de funções nos nós virtuais (*Virtual Boards*) as operações executadas diretamente nos nós *fog*. As funções virtuais podem ser parcialmente ou completamente executadas nos nós *fog*, dependendo das suas capacidades e da complexidade da função. Também, esta camada gerencia a infraestrutura de comunicação (rede) dos nós virtuais e dos nós *fog* por intermédio do componente *Control*

[10]OpenStack - https://www.openstack.org/

*Plane*. O mecanismo de comunicação entre eles é baseado na criação de túneis TCP sobre WebSocket[11]. Assim, esta camada fornece serviços para os desenvolvedores gerenciar e controlar a rede de nós *fog*, bem como virtualizar funções e criar camadas adicionais de sobreposição de rede (do inglês *network overlays*) entre as funções. De acordo com os autores, a camada IaaS da plataforma fornece o serviço chamado de *IoTronic*, que possibilita gerenciar os nós *fog* via linha de comando por intermédio do programa *Stack4Things Command Line Client* e também por uma API Web RESTful. Além disso, o serviço *IoTronic* possui os seguintes agentes: *Registration Agent*, que trata do registro dos nós *fog* ao sistema no momento de inicialização; e *Command Agent*, que trata do gerenciamento e distribuição dos comandos para configuração dos nós além da execução das aplicações.

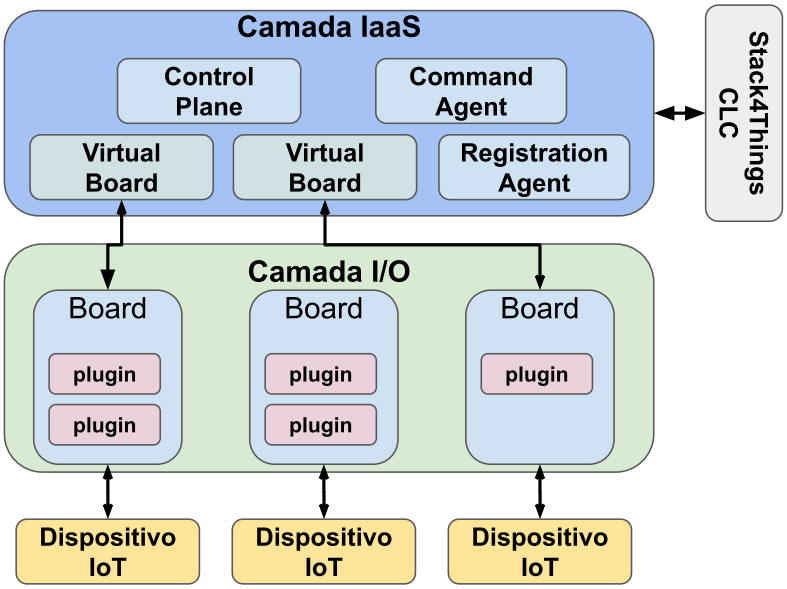



**Figura 1.8. Arquitetura da plataforma Stach4Things (adaptado de [49])**

A **camada I/O** compreende os nós *fog* (representados como *board* na Figura1.8) que estão conectados diretamente aos dispositivos IoT da camada subjacente. A parte da plataforma que é executada nos nós *fog* é chamada de *lightning-rod* e ela é responsável por manter a comunicação com os componentes que estão na nuvem, assim como implementar o protocolo preestabelecido para o registro do nó no sistema. As aplicações são desenvolvidas como *plugins* que são executados nos ambientes de execução mantidos pelos nós *fog*. Os *plugins* são desenvolvidos em código fonte Node.js. Desta forma, o desenvolvedor de uma aplicação projeta e implementa um conjunto de *plugins* para cada aplicação e a plataforma realiza a distribuição destes *plugins* nos diferentes nós considerando suas funções. Por sua vez, o nó mantém o isolamento da execução dos *plugins* de diferentes aplicações fazendo a devida separação dos processos para execução e manutenção do ciclo de vida do código fonte Node.js.

Considerando os requisitos previamente levantados para plataformas *fog computing*, percebemos que a plataforma não apresenta elementos para endereçar os requisitos de suporte a mobilidade, disponibilidade, segurança e privacidade, e eficiência energética.

[11] Protocolo WebSocket - https://tools.ietf.org/html/rfc6455

Assim como na plataforma FogFlow, o requisito escalabilidade também é atendido parcialmente pois a plataforma não oferece mecanismos para escalar automaticamente todo o sistema. Ademais, o desenvolvedor do *plugin* deverá conhecer os detalhes dos dispositivos IoT com que ele vai lidar, de forma a não atender completamente o requisito de heterogeneidade.

### 1.4.3. Microsoft Azure IoT Edge

Microsoft Azure IoT Edge[12] é um projeto *open source* que disponibiliza uma extensão da plataforma Microsoft Azure IoT para as camadas mais próximas dos dispositivos IoT. A plataforma Azure IoT provê um conjunto de componentes modulares, com interfaces bem definidas e que fornecem serviços genéricos relacionados ao domínio IoT como, por exemplo, serviço de ingestão de dados IoT, processamento de *streams*, armazenamento não estruturado de dados IoT, armazenamento de dados em série temporal, *dashboards* e painéis de visualização de dados IoT, serviço de gerenciamento, controle e comunicação com dispositivos IoT, etc. Tais componentes e seus respectivos serviços podem ser usados de diferentes formas dependendo das necessidades das aplicações. Entretanto, todos os componentes da plataforma Azure IoT são executados na nuvem, sendo primordial para o funcionamento existir conectividade entre os dispositivos IoT e a plataforma centrada na nuvem.

A extensão Azure IoT Edge possibilita a execução de alguns componentes da plataforma em nós *fog*, de forma a expor parcialmente as funcionalidades da plataforma próximo à fonte de dados. Desta maneira, uma aplicação é criada utilizando-se múltiplos módulos (*IoT Edge modules*) que são configurados para comunicarem entre si e também com os dispositivos IoT. Os módulos são compatíveis com imagens Docker e são executados em um ambiente de virtualização mantido por cada nó *fog*. Desta forma, o resultado da execução de um módulo pode ser a entrada para outro módulo, de forma a criar um fluxo de processamento (do inglês *pipeline*).

Outra característica da extensão Azure IoT Edge é possibilitar que algumas funcionalidades possam ser executadas nos dispositivos *fog* mesmo sem a conectividade plena com os demais componentes da nuvem. Desta forma, considerando um cenário onde os nós *fog* estão instalados em locais remotos, que não são facilmente acessíveis e com conectividade intermitente, alguns componentes podem ser executados localmente, na borda da rede, e o resultado de sua execução pode ser enviado para a nuvem assim que a conexão for restabelecida. Por outro lado, se o nó *fog* estiver sem conectividade, os componentes da nuvem podem enviar os módulos das aplicações e suas respectivas configurações assim que o nó *fog* ganhar conectividade.

A arquitetura da plataforma está ilustrada na Figura1.9. O componente *Dispositivos* representa os dispositivos IoT com seus sensores e atuadores. Diferentemente da proposta inicial da plataforma Azure IoT, os dispositivos IoT não estão diretamente ligados à nuvem. Em vez disso, eles estão ligados a um nó *fog*, que atua como um *gateway* que pode traduzir os protocolos de comunicação e formatos de dados usados pelos dispositivos IoT. Para cada dispositivo IoT um módulo (*Module*) correspondente deve ser registrado no nó *fog*. Porém, podem existir módulos que não estão associados a dispositivos IoT

[12]Microsoft Azure IoT Edge Project - https://github.com/Azure/iotedge

como, por exemplo, um módulo que implementa uma regra de negócio da aplicação que necessita do resultado do processamento de outros nós que estão diretamente ligados ao dispositivos IoT. Os módulos são orquestrados pelo componente *IoT Edge Runtime*, que por sua vez estabelece a interface de comunicação para os nós se comunicarem entre eles além de gerenciar toda comunicação do nó *fog* com os demais componentes na nuvem. Outra responsabilidade deste componente é reportar a saúde do nó *fog* e dos módulos em execução para a nuvem. Este módulo contém dois subcomponentes, *IoT Edge Agent* e *IoT Edge Hub*, que serão descritos a seguir.

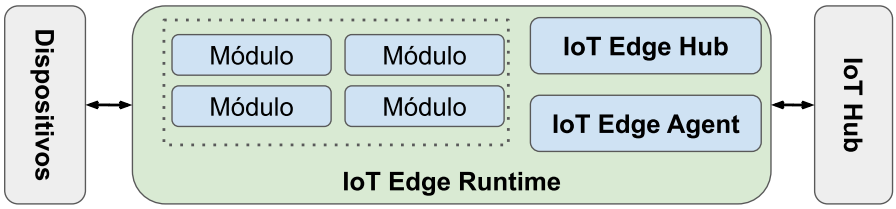



**Figura 1.9. Arquitetura da plataforma Microsoft Azure IoT Edge**

O componente *IoT Edge Agent* é responsável por baixar da nuvem as informações de implantação dos módulos no nó *fog*. Tais informações são encapsuladas em um manifesto de implantação (do inglês *deployment manifest*) mantido pela plataforma Azure IoT na nuvem e que contém, por exemplo, as configurações necessárias para a execução dos módulos, a imagem Docker de cada módulo, as interfaces de comunicação entre os módulos, etc. Desta forma, o componente *IoT Edge Agent* tem a responsabilidade de garantir que o estado e a configuração dos contêineres estejam de acordo com a definição original associada ao nó *fog*. Para tal finalidade, o componente *IoT Edge Agent* deve constantemente sincronizar as mudanças no manifesto mantido na nuvem pela plataforma, iniciando ou finalizando a execução dos módulos no nó *fog*. Em termos de implementação, este componente também é um módulo, sendo sempre o primeiro a ser executado no ato de inicialização do nó *fog*.

Por fim, o componente *IoT Edge Hub* é um *broker* de comunicação que facilita a comunicação local entre os dispositivos IoT e os módulos. O *IoT Edge Hub* atua também como um *proxy* local para o componente *IoT Hub* que está centrado na nuvem. São suportados diversos protocolos de comunicação IoT, como por exemplo AMQP, MQTT e HTTP. Uma característica desta plataforma é a comunicação segura fim-a-fim.

### 1.4.4. ParaDrop

ParaDrop [52, 53] é uma plataforma que permite a execução de aplicações em *gateways* sem fio que estão na borda da rede como, por exemplo, Pontos de Acesso Wi-fi (do inglês *Access Point*) e receptores de TV (do inglês *set-top boxes*). Tais *gateways* são os nós *fog* e os dispositivos IoT estão conectados diretamente a eles. A ideia da plataforma é fazer uso dos recursos computacionais dos *gateways* para executar aplicações que são encapsuladas como imagens Docker. Atualmente os *gateways* conectam os dispositivos da rede, estão sempre ligados e possuem poderosos recursos computacionais que, muitas vezes, são subutilizados. Desta forma, na visão dos autores, utilizar os *gateways* como nós *fog* apresenta alguns benefícios como, por exemplo, a privacidade, uma vez que os

dados sensíveis podem ser processados apenas localmente sem a necessidade de ser enviado para outro local. Também, tal solução pode atender o requisito de baixa latência na comunicação com os dispositivos IoT que estão na mesma rede, além de poder operar mesmo na ausência de conectividade com a Internet.

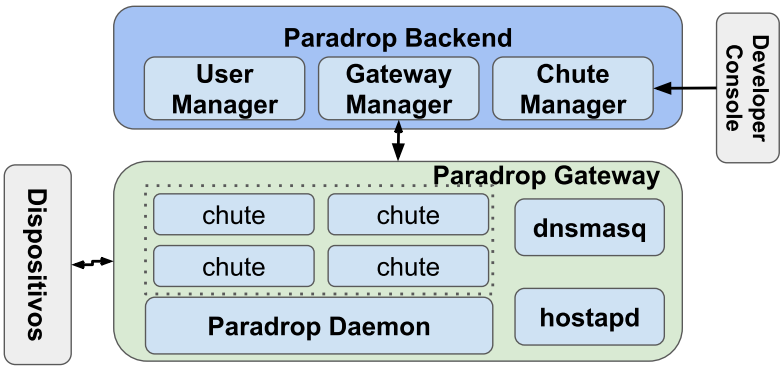



**Figura 1.10. Arquitetura da plataforma ParaDrop (adaptado de [52])**

Conforme ilustrado na Figura 1.10, a arquitetura da plataforma compreende dois elementos: *Paradrop Backend* e *Paradrop Gateway*. A plataforma centraliza a decisão de distribuição das aplicações em um elemento orquestrador na nuvem chamado de *Paradrop Backend*. Os desenvolvedores criam as aplicações, chamadas de *chute*, por intermédio da ferramenta *Developer Console* ofertada pela plataforma. Tais aplicações são encapsuladas no formato de imagens Docker e em um passo posterior, utilizando a mesma ferramenta, os desenvolvedores submetem as aplicações ao componente *Chute Manager*. O componente *Paradrop Daemon*, que está em cada nó *fog*, recebe o pedido de implantação da aplicação e realiza a configuração necessária para o provisionamento de recursos e instanciação da imagem Docker localmente.

O componente *Paradrop Backend* é o elemento centralizado na nuvem que provê o gerenciamento e comunicação entre os desenvolvedores de aplicações e os nós *fog*. A comunicação entre o *Paradrop Backend* e o nó *fog* é via WAMP (*Web Application Messaging Protocol*) enquanto a comunicação entre o *Developer Console* (Desenvolvedor) e o *Paradrop Backend* é por HTTP. O *Paradrop Backend* se comunica com todos os nós *fog* para encaminhar comandos e receber respostas e relatórios do estado de utilização dos recursos em cada nó. Ele também fornece uma interface Web para a fácil visualização dos nós *fog*, registro de usuários (via componente *User Manager*) e instalação de aplicações.

A plataforma ParaDrop requer que em cada nó *fog* um sistema operacional extremamente leve (Ubuntu Snappy), destinado para dispositivos de recursos limitados, seja utilizado. O Ubuntu Snappy fornece um conjunto de ferramentas e APIs para suportar um ambiente de virtualização leve como o Docker. Desta forma, o componente *Paradrop Daemon*, que está presente em cada nó *fog*, tem a responsabilidade de gerenciar o ambiente de virtualização Docker, controlando o uso dos recursos pelas aplicações em execução, além de controlar a comunicação do nó com a parte da plataforma na nuvem (*Paradrop Backend*), reportando periodicamente o estado de utilização dos recursos e recebendo os pedidos para controle das aplicações como, por exemplo, pedidos para iniciar, parar, ins-

talar e desinstalar as aplicações. Também, este componente é responsável por fazer o registro do nó no sistema no ato da inicialização do nó *fog*. Ainda na Figura 1.10, os componentes *hostapd* (*Host access point daemon*) e *dnsmasq* são responsáveis por assegurar a funcionalidade de ponto de acesso do dispositivo usado como nó *fog* e prover serviços de DNS e DHCP.

Ao considerar os requisitos para plataformas *fog computing* previamente apresentados, percebemos que a plataforma não apresenta elementos para endereçar os requisitos de suporte a mobilidade, disponibilidade, ciência de localização e interoperabilidade. A escalabilidade da plataforma é limitada na medida que o orquestrador não faz distinção entre situações onde o nó *fog* está saturado ou não no ato de implantação de uma aplicação. Outro limitante da plataforma é que ela só pode ser usada em alguns poucos tipos de dispositivos *gateways* que apresentam suporte a programação, desta forma não atendendo por completo o requisito de heterogeneidade.

### 1.4.5. Comparação entre as plataformas

Esta subseção apresenta uma comparação entre as plataformas discutidas anteriormente usando os requisitos supracitados (veja seção 1.2), a saber: 1) Distribuição Geográfica; 2) Suporte a aplicações sensíveis a latência; 3) Suporte a Heterogeneidade; 4) Interoperabilidade; 5) Otimização do uso da largura de banda; 6) Ciência de Localização; 7) Ciência de Contexto; 8) Suporte a Mobilidade; 9) Disponibilidade; 10) Eficiência Energética; 11) Segurança e Privacidade; 12) Escalabilidade. Na Tabela 1.1 `sim` significa que a plataforma atende completamente o requisito, `não` representa que a plataforma não atende o requisito, enquanto `parcial` significa que o requisito é atendido parcialmente.

**Tabela 1.1. Plataformas *fog computing* vs Requisitos**

| Requisitos/Plataformas | FogFlow | Stack4Things | Azure IoT Edge | ParaDrop |
|---|---|---|---|---|
| Distribuição Geográfica | sim | sim | sim | sim |
| Latência | parcial | parcial | sim | sim |
| Heterogeneidade | sim | parcial | sim | parcial |
| Interoperabilidade | sim | parcial | parcial | não |
| Largura de banda | sim | sim | sim | sim |
| Ciência de Localização | sim | sim | sim | não |
| Ciência de Contexto | sim | sim | sim | sim |
| Suporte a Mobilidade | não | não | não | não |
| Disponibilidade | parcial | parcial | parcial | não |
| Eficiência Energética | não | não | não | não |
| Segurança e Privacidade | parcial | não | sim | parcial |
| Escalabilidade | parcial | parcial | parcial | não |

Os dados da Tabela 1.1 demonstram que nenhuma plataforma atende a todos os requisitos que estabelecemos anteriormente (Seção 1.2). Os requisitos Distribuição Geográfica, Largura de Banda e Ciência de Contexto são atendidos por todas as plataformas. Já os requisitos de Suporte a Mobilidade e Eficiência Energética não são atendidos por nenhuma plataforma. Ressalta-se que apenas prover a execução das aplicações em dispositivos de baixo poder computacional não é suficiente para prover uma estratégia de

redução de custo de consumo energético em todo o sistema [33]. Por exemplo, na plataforma ParaDrop os nós *fog* estão sempre ligados, independente de existirem aplicações em execução. Também, nenhuma plataforma considerada neste estudo apresenta estratégias para assegurar a Mobilidade de nós *fog*. Tal característica é essencial no contexto de cidades inteligentes onde diversos dispositivos móveis como, por exemplo, telefones celulares e veículos, podem ser usados para processamento e/ou armazenamento de dados.

O requisito Latência não é atendido completamente pelas plataformas FogFlow e Stack4Things devido ao *overhead* dos protocolos que tais plataformas utilizam para a comunicação. Portanto, essas plataformas não são adequadas para aplicações do tipo tempo real (do inglês *real-time*), mas elas podem ser usadas para aplicações próximas a tempo real (do inglês *near real-time*) quanto existe um relaxamento do tempo de resposta da aplicação. O requisito de Heterogeneidade não é atendido completamente pela plataforma Stack4Things pois ela não abstrai os detalhes dos dispositivos IoT e a forma de acessá-los. Já a plataforma ParaDrop pode ser usada em um conjunto limitado de dispositivos que servirão como nós *fog*. O requisitos de Interoperabilidade envolve necessariamente o uso de padrões amplamente aceitos, o que ocorre apenas na plataforma FogFlow.

O requisito Disponibilidade é endereçado parcialmente por quase todas as plataformas, exceto a plataforma ParaDrop. As estratégias usadas para assegurar o fornecimento dos serviços são baseadas no monitoramento constante da saúde dos nós *fog* a fim de que as plataformas possam considerar o estado do nó no processo de orquestração das aplicações. Não foram mencionadas estratégias tradicionais para assegurar disponibilidade como, por exemplo, a adaptação dinâmica da plataforma e migração das aplicações para outros nós. Por fim, o requisito Segurança e Privacidade é atendido plenamente apenas pela plataforma Microsoft Azure IoT Edge enquanto que o requisito Escalabilidade é atendido parcialmente por todas as plataformas.

## 1.5. Estudo de Caso

Esta seção apresenta uma experiência de implementação de uma aplicação IoT usando o FogFlow como plataforma de processamento distribuído na *fog*. Em um primeiro momento, na subseção 1.5.1, será apresentado o processo de instalação e configuração da plataforma em um ambiente distribuído. Este processo é relativamente semelhante ao processo de instalação de outras plataformas *fog computing* que têm os principais componentes tomadores de decisões centrados na nuvem como, por exemplo, o componente de orquestração das tarefas. Logo em seguida, na subseção 1.5.2, será descrita uma visão abstrata da aplicação em termos de tarefas e a ordem de execução destas tarefas. Na subseção 1.5.3 será descrito sucintamente os modelos de programação da plataforma para representar a topologia de uma aplicação, i.e., como a plataforma representa as tarefas de uma aplicação. Na subseção 1.5.4, serão discutidos os operadores que implementam as tarefas e o cenário usado para execução do estudo de caso, apresentando os dispositivos utilizados como nós *fog*. Por fim, na Seção 1.5.5 serão apresentados os resultados obtidos nos testes de execução do estudo de caso, considerando a métrica tempo de processamento demandado pelos operadores que implementam as tarefas da aplicação.

### 1.5.1. Instalação e configuração da plataforma FogFlow

Como já explanado na seção 1.4.1, o FogFlow define a lógica de processamento dos dados através de um DAG de tarefas. Cada vértice deste DAG representa uma tarefa (parte do processamento como um todo) e as arestas definem as interações de entrada e saída de dados entre estas tarefas. Como em um *pipeline* de processamento, o resultado (saída) de uma tarefa é uma entrada para outra tarefa. Cada tarefa é implementada através de um operador na forma de um container Docker. Como um container Docker é dependente da plataforma de *hardware* naqual ele será executado, haverá uma imagem do container do operador para cada tipo de nó *fog* (atualmente são suportadas as arquiteturas ARM e X86). A plataforma faz a orquestração dos operadores nos nós *fog* considerando as capacidades e a localização geográfica do nó em relação a fonte de dados. Os operadores executam o protocolo NGSI e se comunicam por troca de mensagens assíncronas no paradigma *publish/subscribe* assinando e publicando entidades de contexto via *IoT Broker*. O protocolo NGSI determina que as mensagens sejam compatíveis com JSON (*JavaScript Object Notation*) e transmitidas utilizando o protocolo HTTP.

Os componentes da plataforma Fogflow são disponibilizados como imagens Docker e organizados em documento Docker-compose[13], que por sua vez define os parâmetros de configuração para a execução dos contêineres. Esses parâmetros especificam as portas que serão utilizadas por cada componente, bem como as interfaces de rede internas (ou interfaces virtualizadas Docker) de comunicação entre esses componentes. A instalação da plataforma é dividida em duas partes, sendo que cada parte compreende um conjunto de passos que devem ser seguidos em ordem. A Figura 1.11 ilustra as duas partes e segue a nomenclatura proposta pela plataforma. Na parte *cloud* é realizado a instalação dos componentes que tipicamente executam na nuvem. Na parte *Edge* é realizada a instalação dos componentes nos dispositivos que serão usados como nós *fog*. Os componentes da parte *cloud* podem ser instalados em microcomputadores ou microservidores na borda da rede, desde que esses tenham recursos computacionais suficientes e sejam compatíveis com a plataforma X86. Por sua vez, os dispositivos que serão usado como nós *fog* podem ter a arquitetura X86 e ARM e tipicamente possuem recursos computacionais limitados.

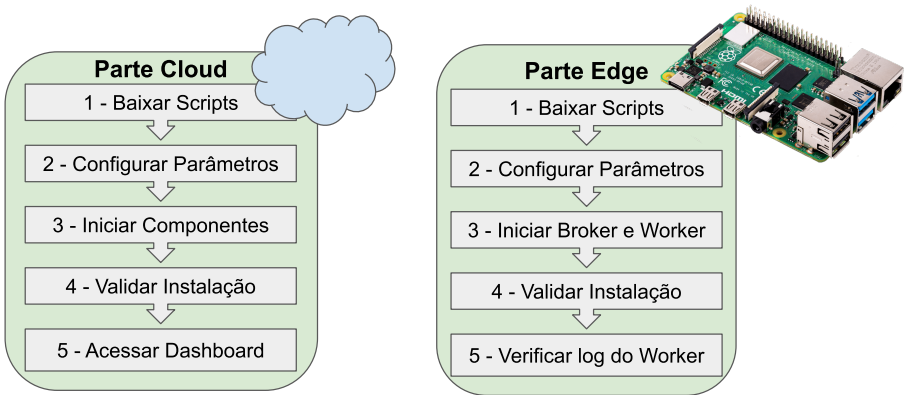



**Figura 1.11. Processo de instalação da plataforma FogFlow**

O primeiro passo na instalação da parte *cloud* envolve o *download* diretamente da

[13]Docker-compose - https://docs.docker.com/compose/

conta GitHub[14] do FogFlow do arquivo Docker-compose e dos arquivos de configuração da plataforma (*config.json*) e do servidor Web Nginx (*nginx.conf*). No arquivo de configuração *config.json* é necessário inserir os endereços IP da máquina que executará a parte *cloud* e da rede interna Docker formada para a comunicação dos componentes da plataforma, além das portas que serão usadas por cada componente. Já o arquivo *nginx.conf* define os parâmetros para a execução do servidor Web utilizado pela plataforma para expor suas funcionalidades. No passo seguinte, Iniciar Componentes, os componentes (imagens Docker) serão inicializados com o Docker-compose (comando `docker-compose up`). O próximo passo, *Validar Instalação*, visa verificar se os componentes foram devidamente inicializados. Basicamente, deve-se verificar se os contêineres foram iniciados (comando `docker ps`) e se a rede interna Docker foi montada de forma apropriada.

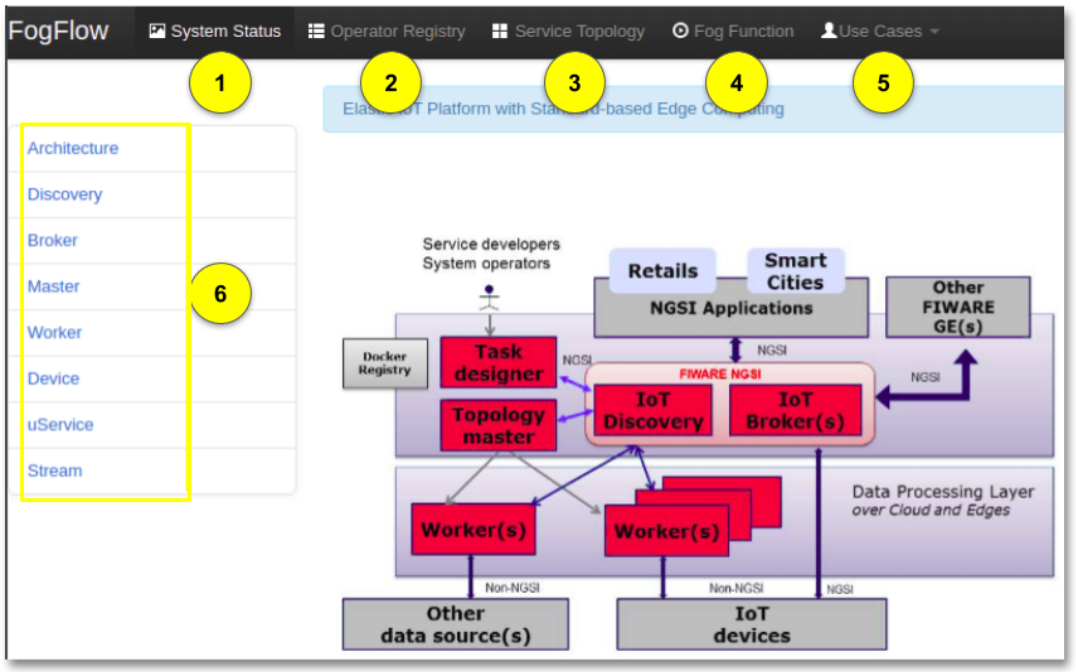



**Figura 1.12. Interface gráfica de gerenciamento da plataforma FogFlow**

A plataforma oferece uma interface gráfica Web, ilustrada na Figura 1.12, para que os desenvolvedores possam especificar as aplicações e verificar o estado dos elementos da plataforma, inclusive a carga de trabalho de cada nó *fog*. Nas abas do painel superior da interface gráfica o usuário tem acesso, respectivamente, ao estado geral da plataforma (círculo 1), a funcionalidade de cadastro de operadores que definem o comportamento das tarefas (círculo 2), a interface para criação de aplicações utilizando-se o modelo de programação *service topology* (círculo 3), a interface para criação de serviços utilizando-se o modelo de programação *fog function* (círculo 4), e alguns estudos de caso que a plataforma fornece (círculo 5). Por fim, na região definida no círculo 6 o usuário tem acesso a algumas informações dos componentes da plataforma e das entidades de contexto que são mantidas pelo *broker*. Por exemplo, são fornecidas informações a respeito da arquitetura da plataforma (*link Architecture*), o endereço (*endpoint*) do componente *IoT*

[14]Conta GitHub do FogFlow - https://github.com/smartfog/fogflow

*Discovery* (*link Discovery*), o endereço do *broker* (*link Broker*), informações da parte *cloud* (*link Master*), informações das aplicações em execução nos nós *fog*, bem como a localização geográfica em um mapa dos nós (*link Worker*), informação dos dispositivos produtores de dados que estão publicando entidades de contexto no *broker* (*link Devices*), e informações das entidades de contexto que são geridas pela plataforma. Os modelos de programação providos pela plataforma, *service topology* e *fog function*, serão descritos mais adiante neste texto.

Na instalação da parte *Edge* deve-se, no primeiro passo, fazer o *download* do *script* de inicialização (*start.sh*) e do arquivo de configuração (*config.json*) dos componentes que executarão nos nós *fog*. O passo seguinte compreende a edição do arquivo *config.json* com a configuração adequada do nó *fog*. Conforme ilustrado na Figura 1.13, no arquivo *config.json* devem ser especificados o endereço IP (atributo `coreservice_ip`) do componente *Orquestrador* (instalado anteriormente na parte *cloud*), o identificador (atributo `site_id`) e as coordenadas geográficas do nó *fog* (atributo `physical_location`), a capacidade (atributo `capacity`) que o nó possui para executar contêineres em paralelo, além de outras configurações. O passo seguinte do processo de instalação consiste na execução do *script start.sh*, que por sua vez configura e executa o contêiner Docker que possui os componentes da parte *Edge*. Por fim, os últimos passos constituem-se da checagem dos contêineres para verificar se a comunicação com os demais componentes distribuídos está ocorrendo adequadamente.

```
{
  "coreservice_ip": "10.7.40.146",
  "external_hostip": "10.7.128.15",
  "internal_hostip": "10.7.128.15",
  "physical_location": {
    "latitude": 35.14703,
    "longitude": 136.786218
  },
  "site_id": "003",
  "logging": {
    "info": "stdout",
    "error": "stdout",
    "protocol": "stdout",
    "debug": "stdout"
  },
  "discovery": {
    "http_port": 80,
    "https_port": 443
  },
  "broker": {
    "http_port": 8070,
    "https_port": 443
  },
  "worker": {
    "container_autoremove": false,
    "start_actual_task": true,
    "capacity": 6
  }
```

**Figura 1.13. Trecho do arquivo de configuração do nó *fog* da plataforma FogFlow**

Os passos executados na parte *edge* devem ser repetidos para cada dispositivo que servirá como nó *fog*.

### 1.5.2. Planejamento Lógico das Tarefas

O cenário do estudo de caso é de processamento de imagens de câmeras de monitoramento, realizado por nós *fog* próximos destas câmeras, atendendo a requisitos de baixa latência e alta vazão de dados (do inglês *throughput*). Para reduzir a latência e aumentar o *throughput*, alguns nós *fog* são equipados com *hardwares* aceleradores com desempenho significativo no processamento de imagens para a detecção e identificação de objetos de interesse. No contexto de um prédio inteligente (do inglês *smart building*), a implementação focou em uma aplicação que permite abrir automaticamente a porta de uma sala quando a câmera instalada na frente desta porta identifica a face (rosto) de uma pessoa como autorizada para tal. As características da faces das pessoas autorizadas foram previamente guardadas em um banco de dados. Assim, a divisão do processamento foi feito com as seguintes tarefas:

- **Face Detection (FD)** - detecta faces no fluxo de vídeo proveniente da câmera gerando uma imagem de interesse para o fluxo de processamento;
- **Face Recognition (FR)** - reconhece as pessoas na imagem de interesse a partir das faces detectadas gerando eventos de interesse;
- **Business Rule (BR)** - emprega a regra de negócio da aplicação no evento de interesse detectado. Nesse caso, a regra de negócio é abrir ou negar a abertura da porta e registrar o evento (detecção de pessoa).

A Figura 1.14 mostra o planejamento lógico das tarefas de processamento. A tarefa FD tem a função de filtrar do fluxo de vídeo da câmera apenas os quadros (imagens) que possuem uma ou mais faces de pessoas, repassando para a tarefa seguinte esta imagem com as coordenadas do(s) retângulo(s) contendo a(s) face(s) detectada(s), bem como a identificação da câmera que gerou tal imagem. Em seguida, a tarefa FR recorta do quadro capturado cada (retângulo de) face e busca em um banco de pessoas previamente conhecidas as faces com as mesmas características, identificando a pessoa daquela face. Por fim, a tarefa BR aplica a ação apropriada para o evento analisado e interage com a aplicação que é executada na nuvem.

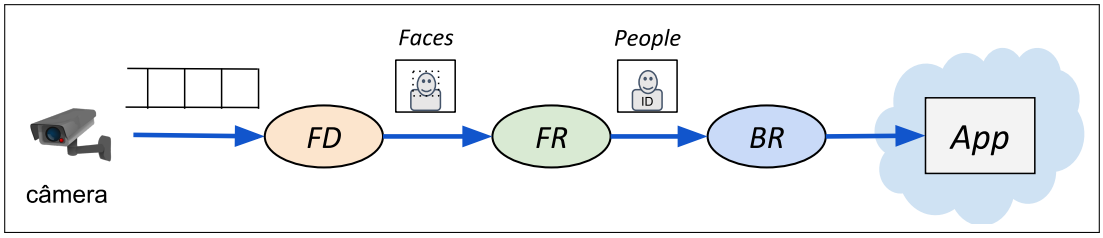



**Figura 1.14. Planejamento lógico das tarefas do estudo de caso**

A Figura 1.15 ilustra os resultados parciais obtidos do fluxo de processamento da imagem de interesse, desde sua obtenção na tarefa FD a partir do fluxo de vídeo, até a identificação da face na imagem de interesse na tarefa FR.

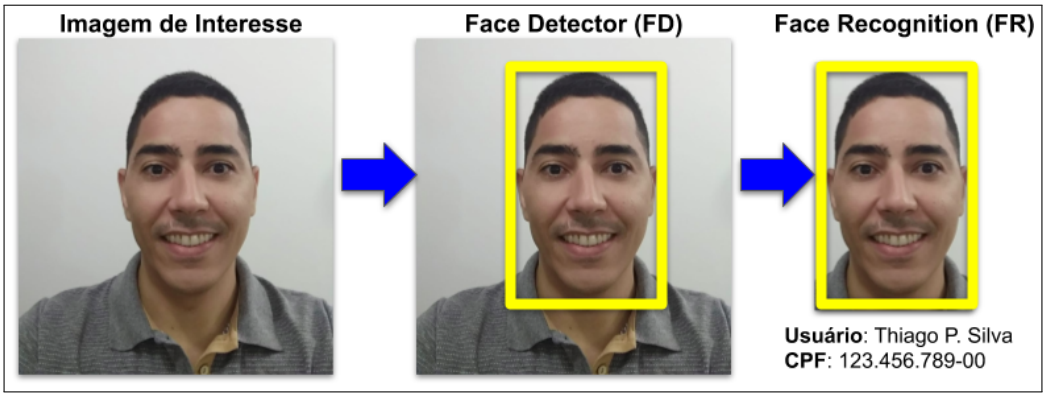



**Figura 1.15. Exemplo dos resultados parciais do fluxo de processamento de uma imagem de interesse**

### 1.5.3. Criação da Topologia de Serviço

No FogFlow, o DAG que representa o fluxo de tarefas de processamento de uma aplicação é chamado de topologia de serviço (do inglês *service topology*). Na topologia do serviço cada tarefa de processamento deve ser implementada através de um operador (do inglês *operator*). Na plataforma, o modelo de programação adotado impõe que cada tarefa que compõe uma aplicação é executada apenas quando surgir no sistema uma entidade de contexto associada à tarefa ou ocorrer uma atualização desta entidade de contexto. Desta forma, os operadores que implementam tais tarefas são tidos como dirigidos por contexto (do inglês *context-driven*) e obrigatoriamente cada operador tem uma entidade de contexto que dispara a sua execução. Por vez, o resultado do processamento de um operador pode ser uma entidade de contexto que dispara a execução de outro operador. Assim, a semelhança da topologia de serviço com um DAG é que os vértices representam as tarefas, com seus respectivos operadores, e as arestas representam as entidades de contexto que determinam a ordem de execução das tarefas.

A plataforma define além do modelo de programação *service topology* outro modelo chamado de *fog function*, que é uma versão simplificada de um topologia de serviço com apenas uma tarefa e, por conseguinte, apenas um operador. Uma *fog function* obrigatoriamente tem uma entidade de contexto que dispara a sua execução e opcionalmente pode gerar como saída do processamento outra entidade de contexto. Neste sentido, uma topologia de serviço pode ser enxergada como um conjunto de *fog functions* que são interrelacionadas pelas entidades de contexto. Desta forma, a plataforma fornece ao desenvolvedor duas maneiras para criar aplicações IoT: a primeira usando *fog function* com a definição da aplicação em apenas uma tarefa; ou usando topologia de serviço com a quebra do processamento várias tarefas.

Uma entidade de contexto, que representa uma unidade de dado a ser processada, é chamada na plataforma de *entity stream*. A tarefa FD é quem inicia o fluxo de processamento, publicando a *entity stream* chamada *Faces*. Uma *entity stream* é compatível com o formato JSON (*JavaScript Object Notation*), conforme ilustrado na Figura 1.16, e obrigatoriamente precisa ter um identificador único (`id`) e um tipo (`Type`). O formato JSON originalmente não permite o transporte de conteúdos binários e, portanto, a imagem de interesse capturada pelo operador FD - que é uma arquivo binário - não é embutida na

*entity stream*. Para contornar esta limitação e não perder desempenho, a estratégia adotada foi manter um servidor HTTP simples e leve que é gerenciado pelo operador FD. Desta forma, as imagens de interesse são obtidas do fluxo de vídeo, armazenadas localmente no dispositivo *fog* que executa o operador FD e compartilhadas pelo servidor HTTP. O atributo `image_url` da *entity stream* contém a URL para que os outros operadores possam recuperar esta imagem. Um outro atributo importante desta entidade de contexto é `faces`, que é um vetor de elementos contendo inicialmente apenas o *bounding box* (coordenadas do retângulo de pixels) envolvendo cada `face` detectada na imagem. Posteriormente serão acrescentados a este vetor faces, pelo operador FR, dados sobre a identificação da face.

```
[
  {
    "entityId": {
      "type": "Devices.Faces.imd-cam-02",
      "isPattern": false,
      "id": "Faces.id"
    },
    "attributes": [
      {
        "name": "image_url",
        "type": "string",
        "value": "http://10.7.16.34.80:88/img/15022272633968.jpg"
      },
      {
        "name": "faces",
        "type": "array",
        "value": [
          {
            "box": [
              358,
              84,
              410,
              160
            ]
          }
        ]
      }
    ]
  }
]
```

**Figura 1.16. Exemplo de conteúdo da entidade de contexto `Faces`**

A definição do fluxo de processamento das *entity streams* é feito através da ferramenta projeto de topologia de serviço (do inglês *designer service topology*) na interface de gerenciamento do FogFlow (círculo 3 da Figura 1.12). Na Figura 1.17, vemos a definição da topologia de serviço (*service topology*) criada para o estudo de caso. Perceba que na representação gráfica os elementos **Task** (em português Tarefa), **Shuffle** e **EntityStream** são representados como retângulos e o fluxo do processamento é definido com as setas que interligam estes elementos. Ademais, esses elementos possuem atributos essenciais utilizados pelo componente *Orquestrador* na decisão de orquestração dos operadores que implementam as tarefas.

No elemento **EntityStream**, o atributo `SelectedType` representa o tipo da entidade de contexto e o atributo `SelectedAttributes` especifica quais atributos da entidade de contexto deverão ser encaminhados para a Tarefa. O atributo `Groupby` especifica a granularidade da entidade de contexto e é usada para definir a quantidade de instâncias do operador relacionado à Tarefa. Por exemplo, considerando uma entidade de contexto do tipo *Pessoa* com o atributo `Groupby` especificado como *EntityID*, então uma instância do operador será executada para cada *id* diferente da entidade de contexto *Pessoa*. Por outro lado, se o atributo `Groupby` for especificado como *ALL*, então apenas uma instância do operador será mantida para todas as entidades de contexto do tipo *Pessoa*, independente do *id* que as entidades possam ter. Já o atributo `Scoped` representa um valor booleano que indica se a informação de localização geográfica está contida na entidade de contexto. Por exemplo, a entidade de contexto que representa uma imagem de interesse capturada por uma câmera, terá o valor do atribuito `Scoped` será setada como Verdadeiro (do inglês *True*), pois a câmera é um sensor produtor de dados fixo que tem sua localização conhecida.

No elemento **Task** o atributo `Name` especifica o nome da tarefa enquanto o atributo `Operator` indica a imagem Docker que implementa tal tarefa. Também, neste elemento, deve ser especificado a saída (do inglês *Output*) gerada pela tarefa. Isto é, qual é a entidade de contexto que será publicada no *broker* após a execução da tarefa, de modo a continuar o fluxo de processamento. Por fim, o elemento **Shuffle** é usada para interligar dois elementos do tipo *Task* (ou em português Tarefa), quando a primeira *Task* produz uma entidade de contexto que é a entrada para a segunda *Task*.

Desta forma, para o estudo de caso foi definida a tarefa *Face_Detection_Task* (FD), que é implementada pelo operador *face_detection_operator*. Este operador inicia o fluxo de processamento do *stream* proveniente da câmera e gera a entidade de contexto *Faces*. Entretanto, devido ao modelo de programação adotado pela plataforma, o início da execução deste operador está atrelado necessariamente ao surgimento da entidade de contexto do tipo *Start*. Desta forma, definimos a entidade de contexto `Start` e atribuímos a propriedade `Groupby` com o valor *EntityID*, de modo que para cada câmera uma instância do operador *face_detection_operator* seja criado.

A tarefa *Face_Recognition_Task* (FR) é implementada pelo operador *face_recognition_operator* e recebe como entidade de contexto o tipo *Faces* e executa o reconhecimento facial, produzindo como resultado a entidade de contexto do tipo *People*, a qual é similar a *Faces* mas contém o atributo `person_id` para cada elemento do vetor do atributo `faces` original. O atributo `person_id` informa uma chave de identificação única de uma pessoa dentro do banco de dados de pessoas conhecidas, como o CPF, por exemplo, ou a palavra *unknown* para indicar que a pessoa não foi localizada neste banco.

Como vimos anteriormente, a comunicação das *entity streams* entre operadores é representada pelo elemento de interface *Shuffle*. Nesse caso, definimos esse agrupamento como *ALL* de forma que todas as entidades de contexto dos tipos *Faces* e *People* serão passados para o próximo operador no fluxo de processamento. A terceira e última tarefa de processamento *Door_Control_Task* é implementada pelo operador *door_control_operator*. Este operador então, ao receber a entidade de contexto `People`, aplica a regra de negócio da aplicação permitindo ou negando a abertura da porta vinculada à câmera. Em

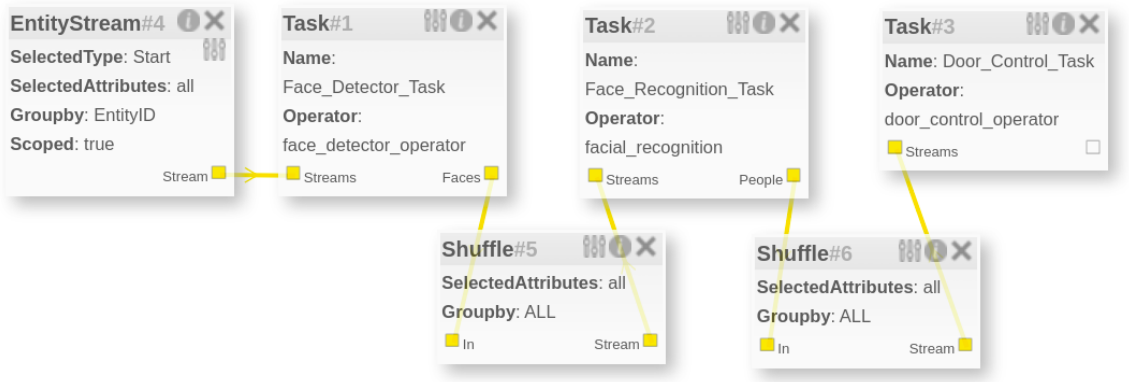



**Figura 1.17. Definição da Topologia de Serviço do estudo de caso**

linhas gerais, *door_control_operator* precisa ter um banco de dados com a identificação das pessoas que estariam autorizadas a entrarem naquela sala associada com aquela câmera/porta, e procurar a pessoa identificada nesse banco. De qualquer forma, a decisão deste operador é notificada ao usuário através de um pequeno *display* na porta, no qual ele vê sua face sendo capturada e, caso necessário, ele aciona a fechadura eletrônica da porta para abri-la. Este banco de dados é encapsulado junto com o código fonte da tarefa no contêiner do operador.

### 1.5.4. Cenário e Configuração dos nós fog usados para o processamento distribuído

Para a execução das tarefas de processamento de imagens preparamos dois dispositivos equipados com aceleradores otimizados para a execução de algoritmos de inteligência artificial, conforme ilustrado na Figura 1.18. O primeiro dispositivo é um Raspberry Pi[15] 4 (CPU ARM quad-core, 4GB de memória RAM) equipado com o acelerador Google Coral[16] Edge TPU (*Tensor Processing Unit*) em uma porta USB-3. O segundo dispositivo é um NVIDIA Jetson Nano[17] (CPU ARM quad-core, 4GB de memória RAM), que possui uma GPU (*Graphics Processing Unit*) NVIDIA Pascal 128-cores já integrada em seu *hardware*. Esse TPU é capaz de executar 4 TOPS (trilhões de operações por segundo) consumindo apenas 2 Watts de energia. Já a Jetson, é capaz de atingir 472 GFLOPS (bilhões de operações com ponto flutuante por segundo) e processar a detecção de objetos em até 8 fluxos de vídeo simultâneos com resolução Full-HD (1920 x 1080 pixels). Os resultados obtidos por esses *hardware* mostram a rápida evolução no desenvolvimento dos dispositivos chamados pela indústria como dispositivos *edge* (do inglês *edge devices*), alcançando capacidade computacional expressiva e que permitem uma certa independência de soluções baseadas na nuvem. Esse aspecto é um dos motivos para o desenvolvimento de aplicações na *Fog*.

O dispositivo Raspberry Pi 4 com seu acelerador Coral ficou com a tarefa *Face_Detection_Task* (FD) no fluxo de vídeo proveniente da câmera. Para esse processamento, foi usado o modelo de rede neural conhecido como MobileNet [54], a qual foi desen-

[15]Raspberry - https://www.raspberrypi.org/

[16]Google Coral - https://coral.ai/

[17]NVIDIA Jetson Nano - https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/


**Figura 1.18. Dispositivos usados como nó *fog*: (a) Raspberry Pi 4 e Google Coral USB e (b) NVIDIA Jetson Nano**

volvida pela Google especialmente para ser usada em dispositivos móveis e embarcados da IoT. Já na Jetson, que ficou com a tarefa *Face_Recognition_Task* (FR), foi usado o modelo de rede neural chamado FaceNet, que aprende um mapeamento de imagens de rosto para um espaço euclidiano compacto, onde as distâncias correspondem diretamente a uma medida de semelhança de rosto [55].

Além destes dois dispositivo, o ambiente ainda contou com um computador laptop onde foram instalados os componentes da parte *cloud* do FogFlow. O operador correspondente a tarefa *Door_Control_Task*, por ter um algoritmo simples e que não faria diferença se fosse executado na *fog* ou na nuvem, foi executado neste computador laptop. A configuração do laptop é um Dell com processador Intel Core i7, 16 GB de memória RAM e placa gráfica dedicada NVIDIA MX150 com 2GB de memória. No cenário montado todos os dispositivos pertencem à mesma rede local.

Não entraremos em detalhes a respeito das técnicas utilizadas na implementação dos operadores que realizam a detecção e identificação das faces, pois estes detalhes estão fora do escopo do minicurso. Entretanto, o código fonte dos operadores está disponível no repositório GitHub[18] e a documentação necessária para criar operadores em linguagem Java, Python e JavaScript pode ser obtida na documentação da plataforma FogFlow[19].

### 1.5.5. Resultados Obtidos

Depois de tudo configurado e instanciado, tanto na parte *cloud* quanto na parte *fog*, iniciamos o experimento de processar o fluxo de vídeo de uma câmera de monitoramento posicionada em frente a uma porta com abertura automatizada. Esta câmera gerava imagens Full-HD (1920 x 1080 pixels) a 25 quadros por segundo com compressão H.264. Foi definida uma *região de interesse* dentro de cada quadro de 500 x 500 pixels representando a posição ideal de captura da face das pessoas. O operador *face_detection_operator* executando no Raspberry Pi 4 acessava o *stream* desta câmera via protocolo RTSP (*Real Time Streaming Protocol*) e passava cada imagem da região de interesse pela rede neural de detecção de faces rodando na TPU (Coral USB). Ao detectar alguma face nesta imagem, *face_detection_operator* publicava a entidade de contexto do tipo *Faces* (re-

[18]Código fonte dos operadores do estudo de caso- https://github.com/thiagopereirasilva/sbrc2020
[19]Documentação da plataforma FogFlow - https://fogflow.readthedocs.io/en/latest/index.html

presentada na Figura 1.16). Em seguida, o operador *face_recognition_operator* recebe a entidade de contexto *Faces*, obtém a imagem JPEG via requisição HTTP ao operador *face_detection_operator*, passa essa imagem pela rede neural rodando em sua GPU para extrair as características da face ali presente, e busca em um banco de dados de faces de pessoas conhecidas, aquelas que são similares à face da imagem. Ao final deste processo, o operador *face_recognition_operator* publica a entidade de contexto do tipo *People* no *broker*. Por último, o operador *door_control_operator* executando no mesmo computador laptop da parte *cloud* do FogFlow, encerrava o processamento gerando a ação de permitir ou negar a abertura da porta.

A Figura 1.19 mostra os tempos que cada operador tomou no processamento do *stream* e na publicação da entidade de contexto resultante do seu processamento. Para o cálculo destes tempos, desenvolvemos um protocolo próprio de contagem de tempo com relógio global rodando em um servidor HTTP na parte *cloud*. O tempo de processamento do primeiro operador *face_detection_operator* foi calculado a partir do instante em que uma pessoa se posiciona na frente da câmera, de forma que a sua face aparece por inteiro na imagem, até o momento de publicar a entidade de contexto do tipo *Faces* no *broker*. O tempo do operador *face_recognition_operator* é contado desde a publicação de *Faces* até este publicar *People*. E por último, o tempo do operador *door_control_operator* é o intervalo entre a publicação de *People* e a tomada de decisão para controlar a porta. Neste gráfico de tempo acumulativo (Figura 1.19), o eixo vertical representa o tempo em segundos necessário para o processamento de cada tarefa pelo operador, enquanto o eixo horizontal representa as iterações executadas. O gráfico ilustra que em todas as 50 interações o tempo total de processamento foi inferior a 0,16 segundos (160 milissegundos) e que o operador *face_detection_operator* foi aquele que demandou o maior tempo de processamento.

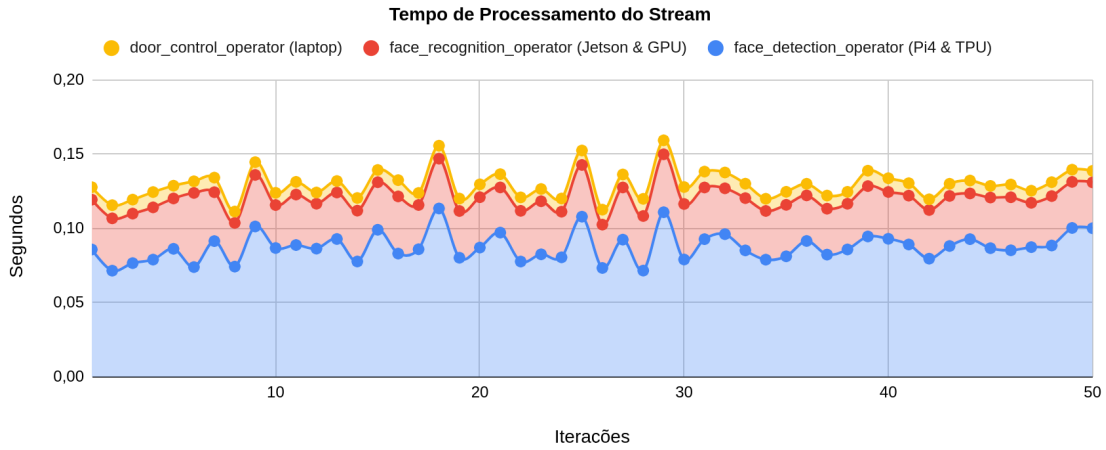



**Figura 1.19. Tempo de processamento dos operadores para o processamento do sream de video**

A Tabela 1.2 mostra os dados estatísticos sobre estes tempos, revelando um bom tempo máximo de processamento, uma vez que também foram computados também todos os tempos de troca de mensagens entre os nós da plataforma. Isto é, não descontamos o tempo necessário para comunicação no *broker*. Dado o tempo médio de 129 milissegundos, se considerarmos que em torno de 1 segundo seja um tempo máximo razoável para

que a pessoa na frente da porta espere esta abrir, poderíamos ter até 8 conjuntos câmera-porta a serem controlados pelo FogFlow com esta topologia e dispositivos *fog*. Além disso, essa quantidade de câmeras pode ser muito maior, pois nesse cálculo da quantidade de câmeras consideramos que a cada 1 segundo haverá uma nova pessoa na frente de cada porta, e isso é uma situação muito rara de acontecer. Ou seja, a verdadeira capacidade de processamento deste experimento em termos de quantidade de câmeras para processar vai depender do volume de eventos que acontecem em cada câmera.

**Tabela 1.2. Estatísticas dos tempos de processamento em segundos por cada operador**

| | **FD** | **FR** | **DC** | **Total** |
|---|---|---|---|---|
| **Média** | 0,087561 | 0,033456 | 0,008885 | 0,129902 |
| **Mínimo** | 0,071595 | 0,028977 | 0,007228 | 0,111474 |
| **Máximo** | 0,113525 | 0,050001 | 0,011576 | 0,159481 |
| **Desvio Padrão** | 0,009567 | 0,003342 | 0,001014 | 0,009792 |

Uma comparação rápida com uma solução cujo processamento do *stream* é feito todo na nuvem, mostra que somente o tempo de transmissão do vídeo para o servidor na nuvem já supera o tempo de processamento na *fog*. Considerando que uma transmissão de vídeo, comprimido em H.264, Full-HD e 20 frames por segundo, gera um *bitrate* de até 30 Mbps, e que a velocidade final entre a câmera e o servidor na nuvem é de 10 Mbps, temos que:

tamanho de um simples quadro = 30 Mbps / 20 fps = 1,5 Mb
tempo de transmissão de um quadro = 1,5 Mb / 10 Mbps = 0,150 segundos

Ou seja, seriam necessários 150 milissegundos para que cada quadro chegasse até o servidor na nuvem, o que já ultrapassa o tempo total de processamento de nosso experimento na *fog*. Portanto, consideramos que, para este tipo de processamento de vídeo monitoramento com IoT, o FogFlow se mostrou uma plataforma bastante promissora.

## 1.6. Conclusão

Este capítulo discorreu sobre plataformas computacionais de *Fog Computing*. Inicialmente foram definidos os requisitos essenciais para uma plataforma de *fog computing* com base em vários trabalhos recentes reportados na literatura da área. Logo em seguida, foram apresentadas duas Arquiteturas de Referência (AR) para *fog computing*, enfatizando a necessidade de padronização para área a fim de proporcionar interoperabilidade entre as plataformas existentes. As duas AR apresentadas são baseadas na arquitetura para *fog computing* proposta por [17] que, na época, os autores vislumbraram *fog computing* como a extensão das capacidades da nuvem para a borda da rede e propuseram uma arquitetura formada por três camadas: i) a camada inferior formada pelos dispositivos IoT que são capazes de sensoriar e atuar no ambiente em questão; ii) a camada intermediária, que é a camada *fog computing*, responsável pelo processamento dos dados localmente e quando da indisponibilidade de processá-los, por encaminhá-los para a nuvem; por fim, iii) a camada superior, formada pelos dispositivos de maior poder computacional na nuvem.

Foram apresentadas algumas plataformas de código aberto, detalhando a organização dos seus componentes e apresentando-se uma análise breve de como atendem (ou não) os requisitos previamente levantados. As plataformas atuais são em sua maioria específicas de domínio e, por conseguinte, possuem diferentes características e atendem também a requisitos inerentes ao domínio onde estão inseridas. A comparação entre as plataformas incluídas neste capítulo revelou que os requisitos de suporte a mobilidade e eficiência energética não são atendidos pelas plataformas. Já o requisito segurança e privacidade é atendido plenamente apenas por uma plataforma enquanto o requisito de disponibilidade é atendido parcialmente pelas plataformas. Desta maneira, percebe-se que muitas questões ainda estão em aberto no paradigma *Fog Computing*, o que torna esta área um campo fértil com várias oportunidades de pesquisa.

Por fim, vimos a viabilidade do paradigma em um estudo de caso para processamento de vídeo em um ambiente distribuído utilizando dispositivos que são coordenados por uma plataforma *fog computing*. Aplicações de análise de vídeo geralmente requerem recursos computacionais robustos para sua execução, uma vez que tais aplicações lidam com uma quantidade grande de dados (*stream* de vídeo). Entretanto, no estudo de caso foi demonstrado o uso de dispositivos de baixo custo e poder computacional para executar tarefas que compõem uma aplicação de processamento de vídeo atendendo os requisitos de baixa latência e redução do uso da largura de banda.

## Referências

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[2] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32, 2014.

[3] Charith Perera, Yongrui Qin, Julio C. Estrella, Stephan Reiff-Marganiec, and Athanasios V. Vasilakos. Fog computing for sustainable smart cities: A survey. *ACM Comput. Surv.*, 50(3):32:1–32:43, June 2017.

[4] Thiago Teixeira, Sara Hachem, Valérie Issarny, and Nikolaos Georgantas. Service oriented middleware for the internet of things: A perspective. In Witold Abramowicz, Ignacio M. Llorente, Mike Surridge, Andrea Zisman, and Julien Vayssière, editors, *Towards a Service-Based Internet*, pages 220–229, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[5] Flavia C. Delicato, Paulo F. Pires, and Thais Batista. *Middleware Solutions for the Internet of Things*. Springer Publishing Company, Incorporated, 2013.

[6] M. Aazam, I. Khan, A. A. Alsaffar, and E. Huh. Cloud of things: Integrating internet of things and cloud computing and the issues involved. In *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014*, pages 414–419, 2014.

[7] Bin Cheng, Guerkan Solmaz, Flavio Cirillo, Ernoe Kovacs, Kazuyuki Terasawa, and Atsushi Kitazawa. FogFlow: Easy Programming of IoT Services Over Cloud and

Edges for Smart Cities. *IEEE INTERNET OF THINGS JOURNAL*, 5(2, SI):696–707, APR 2018.

[8] M. Chiang and T. Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864, 2016.

[9] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog computing for the internet of things: A survey. *ACM Transactions on Internet Technology*, 19, 04 2019.

[10] Larry Feldman (G2) Robert Barton (Cisco) Michael Martin (IBM Canada) Charif Mahmoudi (NIST) Michaela Iorga (NIST), Nedim Goren (NIST). Fog computing conceptual model. *National Institute of Standards and Technology*, 19, 03 2018.

[11] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications*, 98:27 – 42, 2017.

[12] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289 – 330, 2019.

[13] IEEE. *IEEE Std 1934-2018: IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*. IEEE, 2018.

[14] O. Vermesan and P. Friess. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publishers Series in Communications Series. River Publishers, 2013.

[15] E. Marin-Tordera, Xavi Masip, Jordi Garcia Almiñana, Admela Jukan, Guang-Jie Ren, Jiafeng Zhu, and Josep Farre. What is a fog node a tutorial on current concepts towards a common definition, 11 2016.

[16] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. *Fog Computing: A Platform for Internet of Things and Analytics*, pages 169–186. Springer International Publishing, Cham, 2014.

[17] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM. cited By 2351.

[18] Arif Ahmed, HamidReza Arkian, Davaadorj Battulga, Ali J. Fahs, Mozhdeh Farhadi, Dimitrios Giouroukis, Adrien Gougeon, Felipe Oliveira Gutierrez, Guillaume Pierre, Paulo R. R. de Souza, Mulugeta Ayalew Tamiru, and Li Wu. Fog computing applications: Taxonomy and requirements. *ArXiv*, abs/1907.11621, 2019.

[19] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan. Fog computing: Survey of trends, architectures, requirements, and research directions. *IEEE Access*, 6:47980–48009, 2018.

[20] Ranesh Kumar Naha, Saurabh Kumar Garg, and Andrew Chan. Fog computing architecture: Survey and challenges. *ArXiv*, abs/1811.09047, 2018.

[21] Sourav Kunal, Arijit Saha, and Ruhul Amin. An overview of cloud-fog computing: Architectures, applications with security challenges. *Security and Privacy*, 05 2019.

[22] Karima Velasquez, David Perez Abreu, Marcio Assis, Carlos Senna, Diego Aranha, Luiz Fernando Bittencourt, Nuno Laranjeiro, Marilia Curado, Marco Vieira, Edmundo Monteiro, and Edmundo Madeira. Fog orchestration for the internet of everything: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 9, 05 2018.

[23] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. *Fog Computing: A Taxonomy, Survey and Future Directions*, pages 103–130. Springer Singapore, Singapore, 2018.

[24] Md Mahmud and Rajkumar Buyya. *Fog Computing: A Taxonomy, Survey and Future Directions*, chapter 1, page 36. 11 2016.

[25] Ali Fahs and Guillaume Pierre. Proximity-Aware Traffic Routing in Distributed Fog Computing Platforms. In *CCGrid 2019 - IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing*, pages 1–10, Larnaca, Cyprus, May 2019. IEEE.

[26] Lusheng Miao, K. Djouani, B. J. Van Wyk, and Y. Hamam. Performance evaluation of ieee 802.11p mac protocol in vanets safety applications. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1663–1668, 2013.

[27] Anind Dey and Gregory Abowd. Towards a better understanding of context and context-awareness. pages 304–307, 01 2000.

[28] C. Puliafito, E. Mingozzi, and G. Anastasi. Fog computing for the internet of mobile things: Issues and challenges. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6, 2017.

[29] T. Nguyen Gia, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen. Fog computing approach for mobility support in internet-of-things systems. *IEEE Access*, 6:36064–36082, 2018.

[30] Y. Bi, G. Han, C. Lin, Q. Deng, L. Guo, and F. Li. Mobility support for fog computing: An sdn approach. *IEEE Communications Magazine*, 56(5):53–59, 2018.

[31] T. Wang, J. Zhou, A. Liu, M. Z. A. Bhuiyan, G. Wang, and W. Jia. Fog-based computing and storage offloading for data synchronization in iot. *IEEE Internet of Things Journal*, 6(3):4272–4282, 2019.

[32] W. Li, T. Yang, F. C. Delicato, P. F. Pires, Z. Tari, S. U. Khan, and A. Y. Zomaya. On enabling sustainable edge computing with renewable energy resources. *IEEE Communications Magazine*, 56(5):94–101, 2018.

[33] R. Oma, S. Nakamura, T. Enokido, and M. Takizawa. An energy-efficient model of fog and device nodes in iot. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 301–306, 2018.

[34] Gabriel Mujica, Roberto Rodriguez-Zurrunero, Mark Wilby, Jorge Portilla, Ana González, Alvaro Araujo, Teresa Riesgo, and Juan Díaz. Edge and fog computing platform for data fusion of complex heterogeneous sensors. *Sensors*, 18:3630, 10 2018.

[35] Paolo Bellavista, Javier Berrocal, Antonio Corradi, Sajal K. Das, Luca Foschini, and Alessandro Zanni. A survey on fog computing for the internet of things. *Pervasive and Mobile Computing*, 52:71 – 99, 2019.

[36] Upul Jayasinghe, Gyu Myoung Lee, Áine MacDermott, and Woo Seop Rhee. Trustchain: A privacy preserving blockchain with edge computing. *Wireless Communications and Mobile Computing*, 2019:2014697:1–2014697:17, 2019.

[37] M. A. Nadeem and M. A. Saeed. Fog computing: An emerging paradigm. In *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pages 83–86, 2016.

[38] Mohit Taneja and Alan Davy. Resource aware placement of data analytics platform in fog computing. *Procedia Computer Science*, 97:153 – 156, 2016. 2nd International Conference on Cloud Forward: From Distributed to Complete Computing.

[39] A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, and R. Buyya. Chapter 4 - fog computing: principles, architectures, and applications. In Rajkumar Buyya and Amir [Vahid Dastjerdi], editors, *Internet of Things*, pages 61 – 75. Morgan Kaufmann, 2016.

[40] M. Aazam and E. Huh. Fog computing micro datacenter based dynamic resource estimation and pricing model for iot. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 687–694, 2015.

[41] X. Masip-Bruin, E. Marín-Tordera, A. Alonso, and J. Garcia. Fog-to-cloud computing (f2c): The key technology enabler for dependable e-health services deployment. In *2016 Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pages 1–5, 2016.

[42] Amir Sinaeepourfard, John Krogstie, Sobah Abbas Petersen, and Dirk Ahlers. F2c2c-dm: A fog-to-cloudlet-to-cloud data management architecture in smart city. *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 590–595, 2019.

[43] Farhoud Hosseinpour, Yan Meng, Tomi Westerlund, Juha Plosila, Ran Liu, and Hannu Tenhunen. A review on fog computing systems. *International Journal of Advancements in Computing Technology*, 8:48–61, 12 2016.

[44] Wilfried Steiner and Stefan Poledna. Fog computing as enabler for the industrial internet of things. *e & i Elektrotechnik und Informationstechnik*, 133, 10 2016.

[45] Mattia Antonini, Massimo Vecchio, and Fabio Antonelli. Fog computing architectures: A reference for practitioners. *IEEE Internet of Things Magazine*, 2:19–25, 2019.

[46] Elisa Yumi Nakagawa, Flavio Oquendo, and José Carlos Maldonado. *Reference Architectures*, chapter 2, pages 55–82. John Wiley & Sons, Ltd, 2014.

[47] Sourav Kunal, Arijit Saha, and Ruhul Amin. An overview of cloud-fog computing: Architectures, applications with security challenges. *Security and Privacy*, 2(4):e72, 2019.

[48] N. Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–7, 2017.

[49] F. Longo, D. Bruneo, S. Distefano, G. Merlino, and A. Puliafito. Stack4things: An openstack-based framework for iot. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 204–211, 2015.

[50] Giovanni Merlino, Dario Bruneo, Salvatore Distefano, Francesco Longo, and Antonio Puliafito. Stack4things: Integrating iot with openstack in a smart city context. *Proceedings of 2014 International Conference on Smart Computing Workshops, SMARTCOMP Workshops 2014*, pages 21–28, 02 2015.

[51] S. Distefano, G. Merlino, and A. Puliafito. Sensing and actuation as a service: A new development for clouds. In *2012 IEEE 11th International Symposium on Network Computing and Applications*, pages 272–275, 2012.

[52] P. Liu, D. Willis, and S. Banerjee. Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 1–13, 2016.

[53] Suman Banerjee, Peng Liu, Ashish Patro, and Dale Willis. *ParaDrop*, chapter 1, pages 11–23. John Wiley  Sons, Ltd, 2017.

[54] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.

[55] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.

# Chapter 2

# Quantum Internet: The Future of Internetworking

Antônio J. G. Abelém[1], Gayane Vardoyan[2], and Don Towsley[2]

[1]Universidade Federal do Pará - UFPA
Faculdade de Computação – Instituto de Ciências Exatas e Naturais
Av. Augusto Corrêa, 01 – Guamá – Belém – PA – 66075-110

[2]University of Massachusetts - UMass, Amherst
College of Information and Computer Sciences
A327, Lederle Graduate Research Center, Amherst, MA, 01003

***Abstract***

*Quantum information, computation and communication, will have a great impact on our world. One important subfield will be quantum networking and the quantum Internet. The purpose of a quantum Internet is to enable applications that are fundamentally out of reach for the classical Internet. Quantum networks enable new capabilities to communication systems. This allows the parties to generate long distance quantum entanglement, which serves a number of tasks including the generation of multiparty shared secrets whose security relies only on the laws of physics, distributed quantum computing, improved sensing, quantum computing on encrypted data, and secure private-bid auctions. However, quantum signals are fragile, and, in general, cannot be copied or amplified. In order to enable widespread use and application development, it is essential to develop methods that allow quantum protocols to connect to the underlying hardware implementation transparently and to make fast and reactive decisions for generating entanglement in the network to mitigate limited qubit lifetimes. Architectures for large-scale quantum internetworking are in development, paralleling theoretical and experimental work on physical layers and low-level error management and connection technologies. This chapter aims to present the main concepts, challenges, and opportunities for research in quantum information, quantum computing and quantum networking.*

## 2.1. Introduction and Overview

Quantum networks are a critical and highly anticipated component of an ecosystem with a broad spectrum of quantum technologies. This ecosystem will have extraordinary capabilities to effectively solve complex problems in computational sciences, communications, artificial intelligence, and data processing, and will provide a powerful capability for researchers in almost every scientific discipline [European Alliance 2020].

Connecting people or things such as computers, sensors, actuators, or databases that are in separate locations, for technical, economic, political, logistical or sometimes purely historical reasons is the main motivation for building networks, both quantum and classical. What differs is the type of data and operations involved. Quantum computers and quantum networks use quantum information rather than classical information. The analogue of the classical bit is the quantum bit, or qubit for short. Like a classical bit, a qubit has two states, but unlike a classical bit, a qubit may be in a weighted superposition of the two states, allowing certain functions to be evaluated for both input values at the same time [Nielsen 2010].

Quantum communication is a way to transmit signals (either quantum or classical) over distances using the principles of quantum mechanics. Such signals could be used for tasks ranging from cryptography to large-scale distributed quantum computation [Giles 2019]. Quantum communication takes advantage of the laws of quantum physics to protect data and can do this either through the transfer of a quantum state (entangled or not), the creation of an entangled state, or the use of a previously established entangled state. Stephen Wiesner, followed by Charles Bennett and Giles Brassard were the pioneers of modern work on quantum communication [Wiesner 1983]. They explained how to transmit two classical bits of information, while only transmitting one quantum bit from sender to receiver, a result dubbed superdense coding. Bennett and Brassard's proposal [Bennett and Brassard 1984] utilized the new low-level quantum capability of eavesdropping detection to create shared, secret random numbers for keying of classical cryptographic systems. However, quantum key distribution (QKD) in its basic form is limited in distance to a few hundred kilometers in optical fiber or perhaps more through free space using satellites, and is a single-application system.

One important mechanism for transmitting quantum information from one party to another across a geographic distance is teleportation, proposed first by Bennett et al. [Bennett et al. 1993]. Teleportation is the process of using a preshared entangled state between two parties, along with classical communication, to transport a single qubit from one party to another. This process enables a large range of distributed quantum applications. Since teleportation is executed with the help of classical communication, which proceeds no faster than the speed of light, and thus cannot be used for faster-than-light transport or communication of classical information.

Quantum communication consists of either the exchange of quantum information or the sharing of entangled quantum state between two or more parties. This allows the parties to generate long distance quantum entanglement, which serves a number of tasks including the generation of multiparty shared secrets, distributed quantum computing, improved sensing, blind quantum computing, and secure private-bid auctions. However, quantum signals are very fragile, and cannot be copied or amplified. Solu-

tions to these problems are both similar to and different from those for classical networks [Van Meter 2014]. According to Van Meter, all important behaviors of quantum networks arise from dealing with noise and loss using purification and quantum error correction.

One of the major challenges of quantum communication systems lies in the transmission of quantum information with high rates over long distances in the presence of unavoidable losses [Dür 2007]. A solution to this is the introduction of quantum repeaters [Abruzzo et al. 2013], which are an interesting area of research in both experiment and theory. Quantum repeaters enable one to create entangled state between the end points of the network by first dividing the network into segments, creating entanglement across the segments, and then, connecting those entanglements to create the required long range entanglement. In other words, instead of distributing entanglement across a long link, entanglement is generated through smaller links. A combination of entanglement swapping [Zukowski 1993] and entanglement purification [Deutsch 1996] performed at each quantum repeater enables the extension of entanglement across the entire path.

In general, the exchange of data over long distances, in topologically complex networks built on heterogeneous technologies and managed by many independent organizations, requires taking care of noise and loss. In order to enable widespread use and application development, it is essential to develop methods that allow quantum protocols to connect to the underlying hardware implementation transparently and to make fast and reactive decisions for generating entanglement in the network to mitigate limited qubit lifetimes. This can be achieved by a series of layered protocols to provide an abstraction that ultimately allows application protocols to exchange data between two end nodes without having to know any details on how this connection is actually realized. However, only preliminary functional allocation of a quantum network stack has been proposed, and just first versions of physical and link layer protocols have been developed [Dahlberg et al. 2019] [Van Meter 2014].

Quantum Internet has been proposed as the key strategy to significantly scale up the number of qubits for long distance communication of quantum and classical information. However, quantum computing and networking technologies are still at an early stage of research and development (R&D). Architectures for large-scale quantum networking and internetworking are under development, paralleling theoretical and experimental work on physical layers and low-level error management and connection technologies. Exploring how to build it will create opportunities at different levels (service, components, and modules) [S. Wehner and Hanson 2018].

QKD is the best-known application of a quantum Internet. However, there are many other applications that bring advantages that are unattainable with a classical network, such as secure access to remote quantum computers [Kimble 2008], more accurate clock synchronization [Kómár et al. 2014], and scientific applications such as combining light from distant telescopes to improve observations [Gottesman et al. 2012]. Other useful applications will likely be discovered in the next decade, as the development of a quantum Internet progresses.

This chapter aims to present the main concepts, challenges, and opportunities for research in quantum information, quantum computing and quantum networking. Besides this introductory section, this chapter is organized in four more sections. Section 2 pro-

vides a basic understanding of quantum phenomena, such as qubits, superposition, and entanglement. It describes how quantum data is represented and manipulated. Measurement, interference, decoherence, no-signaling and no-cloning theorems and other important concepts are also explained and exemplified. State-vector and Bloch sphere representations and their corresponding manipulations are discussed. The section concludes by presenting Bell-pairs and GHZ states.

Section 3 gives a brief introduction to key quantum communication and quantum networking characteristics. We explain the concepts of teleportation, swapping, how quantum communication channels are implemented, and the new capabilities of quantum networks to communication systems. Differences between quantum and classical networks are also highlighted. How quantum networks deal with noise and loss using purification and quantum error correction will be also discussed. Then, we present QKD, the most important commercial application of quantum communication technology. We conclude the section presenting the groundwork to adapt Internet design principles to the development of quantum networks.

Section 4 presents the current status of the quantum Internet, highlighting the main initiatives, along with challenges, and research opportunities in this emerging area. The focus is on laying the groundwork to adapt Internet design principles towards the development of quantum networks. We discuss the key research challenges and open problems related to the design of a quantum network, which harness quantum phenomena with no counterpart in the classical reality, such as entanglement and superposition, to share quantum states among remote quantum devices.

Section 5 presents the general conclusions of the chapter, as well as a summary of the main contributions of the text. The key strategies of the main countries around the world to advance on the development of foundations for the quantum Internet and significantly scale up the number of qubits for long-distance communication are also presented and discussed.

## 2.2. Quantum Information and Quantum Computing

Until recently, every computer on the planet has operated under rules that Charles Babbage understood and that Alan Turing codified in the 1930s [Turing 1936]. Through the course of the computer revolution, all that has changed at the lowest level are the numbers: speed, amount of RAM and hard disk, number of parallel processors.

Since Turing, quantum computing is the first paradigm that is expected to change the fundamental scaling behavior of algorithms, making certain tasks feasible that had previously been exponentially hard. Of these, the most famous examples are simulating quantum physics and chemistry, and breaking much of the encryption that currently secures the Internet.

In order to understand quantum network operation and importance, we first must learn about the general principles upon which quantum computers are founded. In essence, a quantum computer is a device that takes advantage of quantum mechanical effects to perform certain computations asymptotically faster than a purely classical machine can [Van Meter 2014].

### 2.2.1. Linear algebra and quantum mechanics

First, let us review some basic concepts from linear algebra and describe the standard notation commonly used in quantum computing. Due to page limits and all the content that we will deal with in this chapter, we will not give strict definitions, limiting ourselves to some of the practical questions that the reader needs to know to understand the new concepts that will be presented. A rigorous definition of the linear algebra and quantum mechanics is found in Nielsen's book [Nielsen 2010].

The basic objects of linear algebra are vector spaces. The vector space of most interest to us is $\mathbb{C}^n$, the space of all n-tuples of complex numbers, $(z_1, \ldots, z_n)$. The elements of a vector space are called vectors.

Quantum mechanics is our main motivation for studying linear algebra. Paul Dirac and Erwin Schrödinger played important roles in quantum mechanics. Among other contributions, Dirac introduced the bra-ket notation. A column vector in Dirac's ket notation is represented by the ket $|\psi\rangle$. It is defined as:

$$|\psi\rangle = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix} \tag{1}$$

where $a_i \in \mathbb{C}$, $i = 0, 1, \ldots, N-1$. The bra $\langle\psi|$ is the adjoint (conjugate transpose) of $|\psi\rangle$ given as:

$$\langle\psi| = \begin{pmatrix} a_0^*, a_1^*, \ldots, a_{N-1}^* \end{pmatrix} \tag{2}$$

where $a^*$ is the complex conjugate of $a$. That is, if $a = x + iy$, then $a^* = x - iy$. All vector spaces are assumed to be finite dimensional, unless otherwise noted.

A linear operator between vector spaces $V$ and $W$ is defined to be any function $A : V \to W$ which is linear in its inputs:

$$A\left(\sum_i a_i |v_i\rangle\right) = \sum_i a_i A\left(|v_i\rangle\right) \tag{3}$$

When we say that a linear operator $A$ is defined on a vector space, $V$, we mean that $A$ is a linear operator from $V$ to $V$. An important linear operator on any vector space $V$ is the identity operator, $I_V$, defined by the equation $I_V |v\rangle \equiv |v\rangle$ for all vectors $|v\rangle$. Another important linear operator is the zero operator, which we denote 0. The zero operator maps all vectors to the zero vector, $0|v\rangle \equiv 0$.

An inner product is a function that takes as input two vectors $|v\rangle$ and $|w\rangle$ from a vector space and produces a complex number as output. It can be written as $(|v\rangle, |w\rangle)$, or as $\langle v|w\rangle$. The first notation is commonly used in linear algebra, while the second is the standard quantum mechanical notation. So, we can write the inner product of $|\psi_A\rangle$ and $|\psi_B\rangle$ as:

$$(|\psi_A\rangle, |\psi_B\rangle) = \langle\psi_A|\psi_B\rangle = \sum_{i=0}^{N-1} a_i^* b_i \tag{4}$$

It is important to note that, in the finite dimensional complex vector spaces that come up in quantum computation and quantum information, a Hilbert space [1] is exactly the same thing as an inner product space, that is a vector space $V$ with an inner product on $V$ [Axler 1997].

An eigenvector of a linear operator $A$ on a vector space is a non-zero vector $|v\rangle$ such that $A|v\rangle = \lambda|v\rangle$, where $\lambda$ is a complex number known as the eigenvalue of A corresponding to $|v\rangle$.

Suppose $A$ is any linear operator on a Hilbert space, $V$. It turns out that there exists a unique linear operator $A^\dagger$ on $V$ such that for all vectors $|v\rangle, |w\rangle \in V$,

$$(|v\rangle, A|w\rangle) = \left(A^\dagger|v\rangle, |w\rangle\right) \tag{5}$$

This linear operator is known as the adjoint or Hermitian conjugate of the operator A. From Equation (5) , we can obtain that $(AB)^\dagger = B^\dagger A^\dagger$. By convention, if $|v\rangle$ is a vector, then we define $|v\rangle^\dagger \equiv \langle v|$. With this definition we also see that $(A|v\rangle)^\dagger \equiv \langle v|A^\dagger$. An operator A whose adjoint is A is known as a Hermitian or self-adjoint operator.

A special subclass of Hermitian operators is extremely important, the positive operators. A positive operator on a complex Hilbert space is necessarily a symmetric operator and has a self-adjoint extension that is also a positive operator. A positive operator $A$ is one for which the inner product between $\langle\psi|$ and $A|\psi\rangle$ is greater or equal to 0 (i.e. $\langle\psi|A|\psi\rangle \geq 0$) for all $|\psi\rangle$. A positive definite operator $A$ is one for which $\langle\psi|A|\psi\rangle > 0$ for all $|\psi\rangle \neq 0$.

The notation $U$ will generically be used to denote a unitary operator or matrix. A matrix $U$ is said to be unitary if $U^\dagger U = I$. Similarly an operator $U$ is unitary if $U^\dagger U = I$. So, an operator is unitary if and only if each of its matrix representations is unitary. A unitary operator also satisfies $UU^\dagger = I$.

The trace of a matrix is an important matrix function, very useful in quantum mechanics. The trace of $A$ is the sum of its diagonal elements,

$$tr(A) \equiv \sum_i A_{ii} \tag{6}$$

The trace is cyclic, $tr(AB) = tr(BA)$, and linear, $tr(A+B) = tr(A) + tr(B)$, $tr(zA) = ztr(A)$, where $A$ and $B$ are arbitrary matrices, and $z$ is a complex number. The trace of an operator $A$ is the trace of any matrix representation of $A$ [Nielsen 2010].

An alternate formulation to describe quantum mechanics is to use a tool known as the density operator or density matrix. This alternate formulation is mathematically equivalent to the state vector approach, but it provides a more convenient language for thinking about some commonly encountered scenarios in quantum mechanics. The density operator is used to describe quantum systems whose state is not completely known. Suppose a quantum system is in one of a number of states $|\psi_i\rangle$, where $i$ is an index, with

[1] A Hilbert space is an abstract vector space possessing the structure of an inner product that allows length and angle to be measured.

respective probabilities $p_i$. $\{p_i, |\psi_i\rangle\}$ is usually called an ensemble of pure states. The density operator for the system is formally defined as the outer product of the $|\psi_i\rangle$ and its conjugate $\langle\psi_i|$.

$$\rho \equiv \sum_i p_i \ |\psi_i\rangle \ \langle\psi_i| . \tag{7}$$

The density operator is often known as the density matrix. An operator $\rho$ is the density operator associated with some ensemble $\{p_i, |\psi_i\rangle\}$ if and only if it satisfies the conditions:

1. Trace condition: $\rho$ has trace equal to one.
2. Positivity condition: $\rho$ is a positive operator.

The proof of this theorem and a detailed discussion about density operator is provided by Nielsen et al. [Nielsen 2010].

The notation we review in this section is summarized in Table 2.1:

**Table 2.1. Summary of notations**

| Notation | Description |
|---|---|
| $z^*$ | Complex conjugate of the complex number $z$. |
| $\|\psi\rangle$ | Vector, known as a ket. |
| $\langle\psi\|$ | Vector complex conjugate to $\|\psi\rangle$, known as a bra. |
| $\langle\varphi\|\psi\rangle$ | Inner product between the vectors $\|\varphi\rangle$ and $\|\psi\rangle$. |
| $A^\dagger$ | Hermitian transpose or adjoint of the $A$ matrix. |
| $\langle\varphi\|A\|\psi\rangle$ | Inner product between $\|\varphi\rangle$ and $A\|\psi\rangle$. |
| $tr(A)$ | Trace of a matrix. |
| $\rho$ | Density operator or density matrix. |

### 2.2.2. Quantum bits: qubits

The fundamental concept of classical computation and classical information is the bit. Quantum computation and quantum information are built on an analogous concept, the quantum bit, or qubit. While a classical bit is a data element with two values, 0 and 1, a qubit is represented using either as a true two-level system, such as the polarization of a photon or the spin of an electron, or a pseudo-two-level system, such as two energy levels of an atom that can be treated as a two-level system [Nielsen 2010].

The difference between a classical bit and a qubit is that a qubit can be in a superposition of the two states. The state of a qubit can be written as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{8}$$

where $\alpha$ and $\beta \in \mathbb{C}$ . Put another way, the state of a qubit is a vector in a two-dimensional complex vector space. The special states $|0\rangle$ and $|1\rangle$ are known as computational basis states, and form an orthonormal basis for this vector space.

A classical bit may be examined many times to determine whether it is in the state 0 or 1. However, a qubit generally cannot be examined to determine its full quantum state, that is, the values of $\alpha$ and $\beta$. Instead, quantum mechanics tells us that we can only acquire much more restricted information about the quantum state through a measurement operation. When we measure a qubit we get either the result 0, with probability $|\alpha|^2$, or the result 1, with probability $|\beta|^2$. Naturally, $|\alpha|^2 + |\beta|^2 = 1$, since the probabilities must sum to one. Geometrically, we can interpret this as the condition that the qubit's state be normalized to length 1. Thus, in general a qubit's state is a unit vector in a two-dimensional complex vector space.

As $|\alpha|^2 + |\beta|^2 = 1$, $|\psi\rangle$ can also be expressed as:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle. \tag{9}$$

This equation facilitates the representation of a qubit's state in a three-dimensional sphere, called *Bloch sphere*, as shown in Figure 2.1. The numbers $\theta$ and $\varphi$ define a point on the unit three-dimensional sphere. The south-north axis is the Z-axis, the positive X-axis is toward the reader (out of the page or screen) and the Y-axis is right-left.

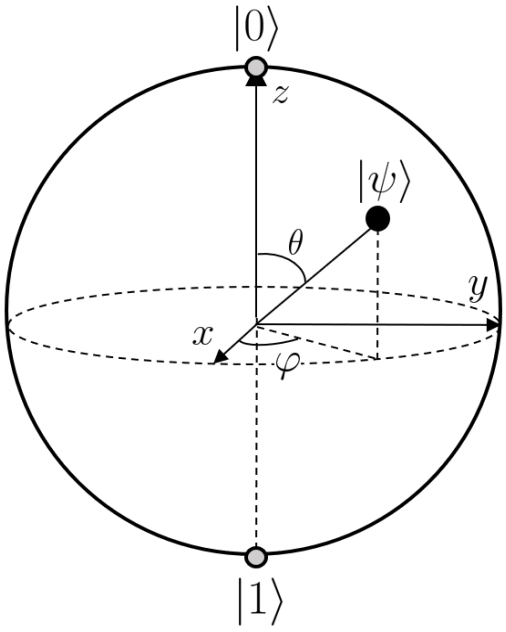



**Figure 2.1. Bloch sphere**

The *Bloch sphere* provides a useful means of visualizing the state of a single qubit, and often serves as an excellent testbed for ideas about quantum computation and quantum information. If the vector points at the north pole, our qubit is in the $|0\rangle$ state, and if it points at the south pole, the qubit is in the $|1\rangle$ state. When the unit vector points toward you, that is the $(|0\rangle + |1\rangle)/\sqrt{2}$ state; when it points away from you, that is the $(|0\rangle - |1\rangle)/\sqrt{2}$ state. These two states are called the $|+\rangle$and $|-\rangle$ (read "ket plus" and "ket minus") states. The positive Y-axis is $(|0\rangle + i|1\rangle)/\sqrt{2}$ and the negative Y-axis is $(|0\rangle - i|1\rangle)/\sqrt{2}$. The phase is the position of the vector about the Z-axis.

### 2.2.3. Multiple qubits

While the state vector is two-dimensional for a single qubit, the state vector is $N = 2^n$ dimensional for a n-qubit register. Suppose we have a two qubit system, it will have four computational basis states denoted $|00\rangle, |01\rangle, |10\rangle, |11\rangle$. A related set of two or more qubits is commonly referred to as a quantum register [Hagouel and Karafyllidis 2012].

A pair of qubits can also be in superpositions of these four states. In this way, the state vector describing the two qubits is

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle, \tag{10}$$

where each computational basis state is associated with a complex coefficient, called an amplitude.

An important two qubit state is the Bell state or EPR pair

$$(|00\rangle + |11\rangle)/\sqrt{2}. \tag{11}$$

It is the key ingredient in quantum teleportation, which forms the foundation of much of quantum networking, as we will see in Section 2.3.

Bell pairs have the property that the qubits are correlated or entangled. Quantum entanglement is a quantum mechanical phenomenon in which the quantum states of two or more objects have to be described with reference to each other, even though the individual objects may be spatially separated. These correlations have been the subject of intense interest ever since a famous paper by Einstein, Podolsky, and Rosen [Einstein et al. 1935]. In the 1960s, John Bell extending and clarifying the work of Einstein et al. and proved that the measurement correlations in the Bell state are stronger than could ever exist between classical systems [Bell and Aspect 2004].

If we generalize and consider a system of n qubits, the computational basis states of this system are of the form $|x_1\, x_2\, x_3\, \ldots\, x_n\rangle$, and a quantum state of such a system is specified by $2^n$ amplitudes. We use the tensor product to compose the state of two or more qubits into one vector, or operations on multiple qubits into a single operator [Bourbaki 1989].

### 2.2.4. Quantum gates and quantum circuits

Quantum computation proceeds by taking a set of qubits, modifying their states such that a "computation" of some interest is performed and reading out the result so that we learn what happened. Analogous to the way a classical computer is built from an electrical circuit containing wires and logic gates, a quantum computer is built from a quantum circuit containing wires and elementary quantum gates to carry around and manipulate the quantum information [Nielsen 2010].

In the circuit model, quantum computations are decomposed into separate gates and can be organized more or less along the lines of classical circuits. In order for our computational capabilities to be "universal", we must be able to reach any point on the Bloch sphere for a single qubit.

Consider, for example, classical single bit *NOT* gate $(X)$, whose operation is defined by its truth table, in which $0 \rightarrow 1$ and $1 \rightarrow 0$, that is, the 0 and 1 states are interchanged. In the quantum world, a single-qubit operation can be any rotation on the Bloch sphere. Rotations about the axes of the Bloch sphere can be described in terms of the Pauli matrices, which are a set of three 2 x 2 complex matrices which are Hermitian and unitary, that arise in Pauli's treatment of spin in quantum mechanics [Nielsen 2010]. However, specifying the action of the gate on the states $|0\rangle$ and $|1\rangle$ does not tell us what happens to superpositions of the states $|0\rangle$ and $|1\rangle$, without further knowledge about the properties of quantum gates. In fact, the quantum *NOT* gate acts linearly, that is, if $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, then $X|\psi\rangle = \alpha|1\rangle + \beta|0\rangle$

Unlike the classical case where there is only one nontrivial gate (*NOT* gate), there are many non-trivial single-qubit gates. Two important ones are the Z gate and the Hadamard gate $(H)$. The first one leaves $|0\rangle$ unchanged, and flips the sign of $|1\rangle$ to give $-|1\rangle$.

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{12}$$

The second one, the Hadamard gate has representation:

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{13}$$

when applied to $|0\rangle$, it returns $H|0\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$, while when applied to $|1\rangle$, it returns $H|1\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$.

The Hadamard gate is one of the most useful quantum gates and is worth visualizing its operation on the Bloch sphere as illustrated in Figure 2.2. Geometrically, we visualize the Hadamard operation as a $90^o$ rotation about the Y-axis, followed by a $180^o$ rotation about the X-axis [Nielsen 2010]. Figure 2.3 summarizes single qubit gates presented.

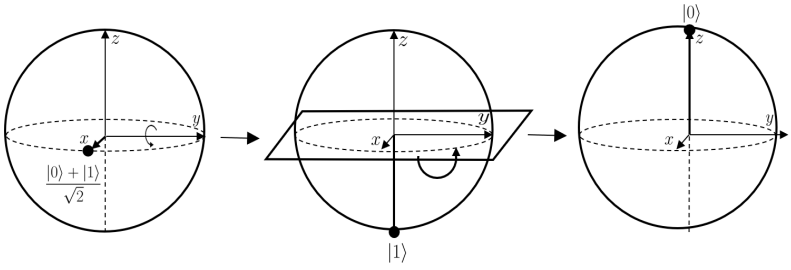



**Figure 2.2. Hadamard gate on the Bloch sphere, acting on the input state** $(|0\rangle + |1\rangle)/\sqrt{2}$

As computations involve more than one qubit, let us generalize from one to multiple qubits. First, consider the controlled-NOT gate, or CNOT. This gate has two input qubits, known as the control qubit and the target qubit, respectively. If the control qubit is one, a NOT operation is performed on the target qubit; if the control qubit is zero, the

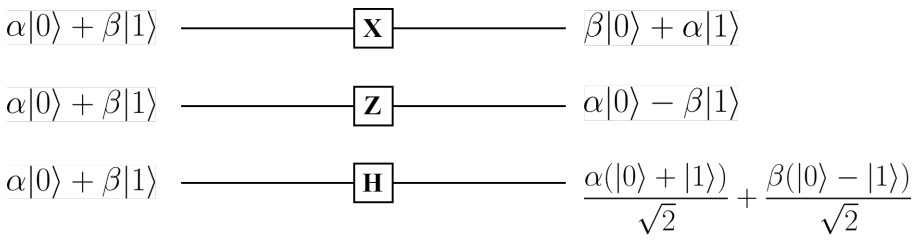



**Figure 2.3. More important single qubit gates**

target bit is left unchanged. The output is the exclusive OR (XOR) of the two qubits, and one of the input qubit and may be summarized as $|A,B\rangle \rightarrow |A,A \oplus B\rangle$. Table 2.2 shows the truth table for a CNOT with A as the control bit and B as the target bit.

**Table 2.2. Controlled-NOT truth table**

| Input | Output |
|---|---|
| AB | AB |
| $\|00\rangle$ | $\|00\rangle$ |
| $\|01\rangle$ | $\|01\rangle$ |
| $\|10\rangle$ | $\|11\rangle$ |
| $\|11\rangle$ | $\|10\rangle$ |

The circuit representation for the *CNOT* is shown in the Figure 2.4. The top line represents the control qubit, while the bottom line represents the target qubit.

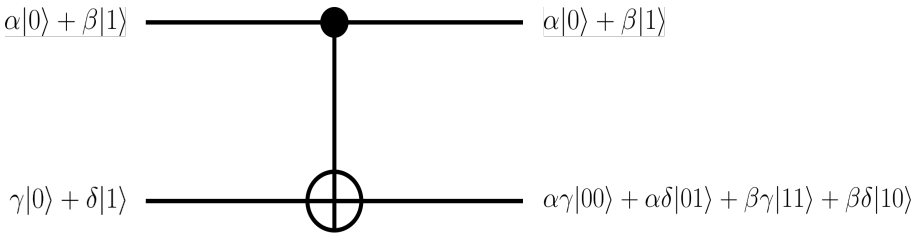



**Figure 2.4. Controlled-NOT gate**

A quantum computation, in the abstract, is a unitary transformation on an initial quantum state, creating desired states, which we can then measure. A complete unitary transform on $n$ qubits, of course, is a $2^n$ $x$ $2^n$ matrix; therefore, direct construction of the unitary to implement a complex function of more than a few qubits is difficult. A quantum circuit effects the overall transform via a series of smaller gates (generally, one to three-qubit gates) applied in a prescribed order on the appropriate qubits.

Researchers have found several methods for decomposing a specific unitary transform into a series of small gates or operations that we know how to implement. Figure 2.5 shows a simple example of a four qubit quantum circuit. This circuit consists of two Hadamard gates and three CNOT gates. Gates on different qubits can be executed at the same time, as shown by the vertical alignment.

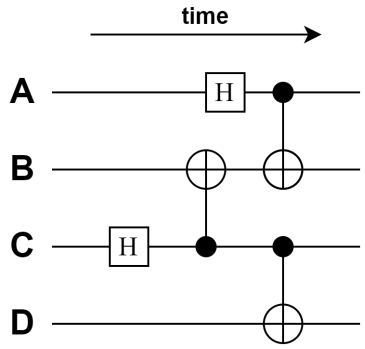



**Figure 2.5. Simple quantum circuit**

### 2.2.5. Measurement

In quantum physics, measurement is the testing or manipulation of a physical system in order to yield a numerical result. The predictions that quantum physics makes are in general probabilistic [Holevo 2001].

As presented in Section 2.2.2, a qubit described by the Equation (8) can exist in a continuum of states between $|0\rangle$ and $|1\rangle$. When a qubit is measured, it only ever gives $'0'$ or $'1'$ as the measurement result - probabilistically. The measurement changes the state of a qubit, collapsing it from its superposition of $|0\rangle$ and $|1\rangle$ to the specific state consistent with the measurement result. For example, consider that a qubit is in the state $|+\rangle$

$$(|0\rangle + |1\rangle)/\sqrt{2}. \tag{14}$$

If measurement gives 0, then the post-measurement state of the qubit will be $|0\rangle$.

In an analogous way it is possible in principle to measure a quantum system of many qubits with respect to an arbitrary orthonormal basis. For two or more qubits, we can measure either the entire system or only part. Measuring a single qubit can alter the state of the system. For example, consider the state vector of the Equation (10), describing the two qubit in superpositions. The measurement result $x(= 00,\ 01,\ 10\ or\ 11)$ occurs with probability $|\alpha_x|^2$, with the state of the qubits after the measurement being $|x\rangle$, in a similar way to the case for a single qubit.

As we mentioned in Section 2.2.3, the Bell state has the important property that the measurement outcomes are entangled. Upon measuring the first qubit, one obtains two possible results: 0 with probability $1/2$, leaving the post-measurement state $= |00\rangle$, and 1 with probability $1/2$, leaving the post-measurement state $= |11\rangle$. As a result, a measurement of the second qubit always gives the same result as the measurement of the first qubit. In others words, measuring one qubit has determined the state of the other.

In the literature, the measurement operation is represented by a 'meter' symbol, as shown in Figure 2.6. The input is a qubit in a state $|\psi\rangle$ and the output is a classical bit, distinguished from a qubit by drawing it as a double-line wire. For readers interested in this fascinating topic, one good place to start studying is Preskill's lecture notes [J. 1998].

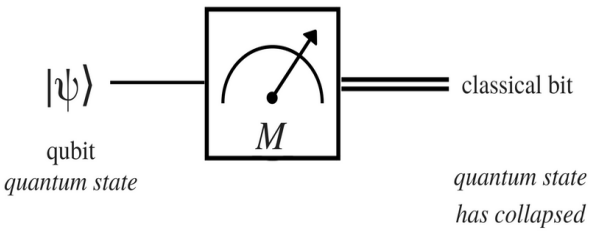



**Figure 2.6. symbol for measurement.**

### 2.2.6. Interference, decoherence and fidelity

In physics, interference is the combination of two or more waveforms to form a resultant wave, in which the displacement is either reinforced or canceled [Steel 1986]. Quantum interference can happen between particles that arrive at the same position or quantum state but by different paths. Quantum interference, a byproduct of superposition, is what allows us to bias the measurement of a qubit toward a desired state or set of states.

Decoherence is the gradual decay of the state of a system. It happens because quantum states are very fragile: excited atoms decay and spins of electrons and atomic nuclei spontaneously flip. Any quantum system is affected through interactions with its environment, leaking information about its state out into the environment where it cannot be recovered.

When decoherence occurs, measurement of the system may not produce the desired results, causing the failure of our quantum algorithm. The two key measures of decoherence are the T1 and T2 times. T1 is the energy relaxation time, and T2 is the phase relaxation time. Both processes are memoryless, with probabilistic behavior. The amount of time we can count on the state of a qubit remaining in a usable state is a function of the minimum of T1 and T2. Researchers determine these values experimentally, and an important area of device research is extending these times by careful engineering of the environment and control system.

Fidelity is used to track the quality of the state. Fidelity ranges from 0 to 1.0, with the latter being perfect. It is, essentially, the probability that our qubit or set of qubits is actually in the state we believe it ought to be in. In other words, fidelity of a state corresponds to how imperfect it is in relation to a desired state [Jozsa 1994, Liang et al. 2019]. It is not a metric, but has some useful properties and it can be used to define a metric on this space of density matrices. We will define the fidelity as

$$F = \langle \psi | \rho | \psi \rangle \tag{15}$$

where $0 \leq F \leq 1$ is the fidelity [2], $|\psi\rangle$ is the state we think we have created and $\rho$ is the density matrix of the actual state.

The fidelity can also be thought of the overlap of our actual state with the desired

[2]In literature, the fidelity is often defined as $F = \sqrt{\langle \psi | \rho | \psi \rangle}$, but in keeping with Jozsa's definition [Jozsa 1994], adopted also in Van Meter's book [Van Meter 2014], we dispense with the square root.

state. The fidelity is 1.0 for a pure state and declines as noise in the system degrades the quality of the state. Consider, for example, we are initializing a two-qubit register to the $\psi = |00\rangle$ state, but that the initialization process is imperfect. To learn how imperfect, we repeat the process a number of times and measure the state, to build up a statistical picture of our ability to create the desired state. From this process, we obtain the density matrix $\rho_\psi$ and, then, we can calculate the fidelity $F_\psi$ for our desired state. For an $n$-qubit state, the completely mixed state in which all qubits are random, we have $F = 2^{-n}$.

### 2.2.7. Bell pairs and GHZ states

As we mentioned in Section 2.2.3, the best known two qubit entanglement involving two parts sharing two qubits is the Bell state. In the addition to the one listed in $(11)$ there are three others forms of Bell pairs:

$$(|00\rangle - |11\rangle)/\sqrt{2} \tag{16}$$

$$(|01\rangle + |10\rangle)/\sqrt{2} \tag{17}$$

$$(|01\rangle - |10\rangle)/\sqrt{2} \tag{18}$$

With Bell pairs $(11)$ and $(16)$, a measurement of one qubit will result in both qubits being zero or both qubits being one, with equal probability. For example, Alice may hold one qubit, while Bob holds another one, at an arbitrary distance apart without the behavior of the Bell pair changing. When Alice measures her qubit and finds a one, she will be sure that when Bob measures his qubit, it will be a one. Likewise, if she measures zero, Bob will measure zero. In $(17)$ and $(18)$, in contrast, if Alice measures a one, Bob will measure a zero and vice-versa. Moreover, this effect does not change if Bob measures his qubit first or if they both measure their qubits simultaneously.

There are other, larger multi-party entangled states that are useful for a variety of tasks, the Greenberger-Horne-Zeilinger (GHZ) state [Bravyi et al. 2006]. It was first studied by Daniel Greenberger, Michael Horne and Anton Zeilinger in 1989. It involves at least three subsystems (particle states, or qubits) and allows to observe extremely non-classical properties of the state [Greenberger et al. 2007].

The GHZ state is an entangled quantum state of $M > 2$ subsystems. In simple words, it is a quantum superposition of all subsystems being in state 0 or all of them being in state 1, represented by:

$$\frac{|000...\rangle + |111...\rangle}{\sqrt{2}} \tag{19}$$

There is no standard measure of multi-partite entanglement because different, not mutually convertible, types of multi-partite entanglement exist. Nonetheless, many measures define the GHZ state to be maximally entangled state.

GHZ states are used in several protocols in quantum communication and cryptography, including any of several common forms of three-party or larger states, for example, in secret sharing or in the Quantum Byzantine Agreement [Van Meter 2014].

## 2.3. Quantum Communication and Quantum Networks

Quantum communication consists of either the exchange of quantum information or the sharing of entangled quantum state between two or more parties. It takes advantage of the laws of quantum physics to protect data and offers new functionality over classical communication. Its most interesting application is protecting information channels against eavesdropping by means of quantum cryptography.

To transport qubits from one node to another, we need communication Channels. Quantum communication channels are implemented by sending states of light down a physical channel. These states may be single photons or other quantum optical states with either large or small numbers of photons. For the purpose of quantum communication, standard telecom fibers can be used or free space. It may involve a single transmitter and receiver, or multiple receivers that can individually be enabled or disabled in a shared bus configuration. A link uses a quantum channel and associated classical channel to connect two or more nodes.

To make maximum use of communication infrastructure, we also require optical switches capable of delivering qubits to the intended quantum processor. These switches need to preserve quantum coherence, which make them more challenging to realize than standard optical switches. Current commercial switches have various problems that make them unsuitable for rerouting entangled photons. Those that are made of micro-electromechanical components keep entangled states intact but operate too slowly. Other optoelectronic switches either add too much noise so that single photons are difficult to detect, or they completely destroy the quantum information. The utilization of a quantum switch provides significant advantages for a number of problems, ranging from quantum computation and quantum information processing, through non-local games to quantum communication [Caleffi 2020].

Quantum networks enable the secure transmission and exchange of quantum communications between distinct, physically separated quantum processors, or endpoints. However, quantum signals are weak and very fragile and cannot be copied or amplified. Consequently, quantum operations are needed to exchange data over long distances to deal with noise and loss.

This section provides a brief introduction to quantum communication and networking. We present the concept of teleportation, swapping, and how quantum communication channels are implemented. Differences between quantum and classical networks are also highlighted. How quantum networks deal with noise and loss using purification and quantum error correction will be also discussed. Then, we present the most important commercial application of quantum communication technology, Quantum Key Distribution (QKD). We conclude the section presenting the groundwork to adapt Internet design principles to the development of quantum networks.

### 2.3.1. Quantum teleportation

As mentioned in Section 2.1, quantum teleportation is the process by which quantum information can be transmitted from one location to another, with the help of classical communication and previously shared quantum entanglement between two parties.

Teleportation is best described through the communication between Alice and Bob. Suppose Alice and Bob generated a Bell pair, each taking one qubit of the Bell pair. Alice must deliver a qubit $|\psi\rangle$ to Bob. She does not know the state of the qubit, and can only send classical information to Bob. Alice interacts the qubit $|\psi\rangle$ with her half of the EPR pair, and then measures the two qubits in her possession, obtaining one of four possible classical results, 00, 01, 10, and 11. She sends this information to Bob. Depending on Alice's classical message, Bob performs one of four operations on his half of the EPR pair. By doing this he can recover the original state $|\psi\rangle$.

Teleportation consumes exactly one Bell pair. Two classical bits of measurement result must be communicated. Hence, the time to execute one operation corresponds to the time needed to transmit the classical information. Careful engineering may allow pipelining of this operation with others [Bennett et al. 1993].

Figure 2.7 shows the quantum circuit that implements the teleportation of a qubit. The state to be teleported is $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha$ and $\beta$ are unknown amplitudes. The state input into the circuit is $|\psi_0\rangle = |\psi\rangle|\beta_{00}\rangle$. The bottom line represents Bob's system, while the two top lines are Alice's system. The single lines denote qubits, the meters represent measurement operations, and the double lines coming out of them carry represent classical bits.

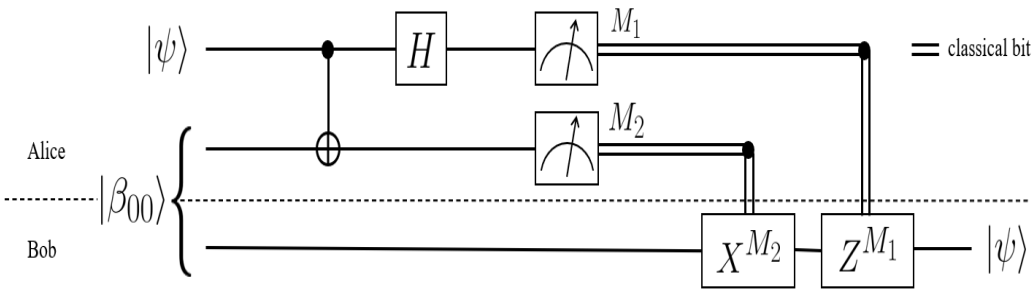



**Figure 2.7. Quantum circuit for teleporting a qubit**

There are two interesting features of teleportation. Quantum teleportation does not enable faster than light communication, because to complete the teleportation Alice must transmit her measurement result to Bob over a classical communication channel. Second, teleportation doesn't violate the no-cloning theorem, creating a copy of the quantum state being teleported. This violation is only illusory since after the teleportation process only the target qubit is left in the state $|\psi\rangle$, and the original data qubit ends up in one of the computational basis states $|0\rangle$ or $|1\rangle$, depending upon the measurement result on the first qubit.

Teleportation depends on the ability to create entangled Bell pairs over some distance. Naturally, many of the experimental groups involved in teleportation have also pushed the boundaries of what is possible to create larger, longer-distance, higher-fidelity, or longer-lived entangled states. In 1998, Boschi et al. verified the initial boundaries of teleportation [Boschi et al. 1998]. The distance was increased in August 2004 to 600 me-

ters, using optical fiber [Ursin et al. 2004]. Subsequently, the record distance for quantum teleportation has been gradually increased to 16 kilometers (9.9 mi) [Jin et al. 2010], then to 97 km (60 mi), and after to 143 km (89 mi), set in open-air experiments in the Canary Islands, done between the two astronomical observatories of the Instituto de Astrofísica de Canarias [Ma et al. 2012]. Takesue et al. reached the distance of 102 km (63 mi) over optical fiber in 2015 using superconducting nanowire detectors [Takesue et al. 2015]. The group of Jian-Wei Pan [Ren et al. 2017] reported the distance of 1,400 km (870 mi) by using satellite for space-based quantum teleportation, point out need to distribute Bell pairs across distances motivated in part by teleportation.

### 2.3.2. Entanglement swapping

A simple and illustrative example of teleportation is the entanglement swapping, which will be seen to be very important for networking. The term "entanglement swapping" was introduced by Zukowski et al. [Zukowski 1993], originally in the context of photonic Bell pairs created via PDC and coupled using beamsplitters. However, the entanglement swapping can be used to distribute Bell pairs across long distance by teleporting the state of one member of a Bell pair over progressively longer distances until the pair stretches from end to end.

Suppose Alice has a particle which is entangled with a particle owned by Bob, and Bob teleports it to Carol, then afterward, Alice's particle is entangled with Carol's. Figure 2.8 illustrates the process. Bob holds one end of each of two Bell pairs, one coupled to a qubit with Alice, the other to a qubit with Carol, which we will call $|\psi^-\rangle^{(AB)}$ and $|\psi^-\rangle^{(BC)}$ respectively. Bob decides to lengthen the pair on the left using the pair on the right. The results of this operation must be communicated to Carol, allowing Carol to recreate the state of resulting in a new Bell pair.

In theory, Alice never needs to be told that the operation has occurred. Although Carol must apply corrective operations to complete the reconstruction, Alice is entirely passive, merely storing its half of the Bell pair in a buffer memory. However, Alice is very likely waiting on the completion of the swapping operation in order to perform some other action; at the very least, an application at node Alice is waiting to use the end-to-end Bell pair.

### 2.3.3. Purification and error correction

According to Van Meter, purification is the process of improving our knowledge about the state, by testing propositions about it [Van Meter 2014]. This improvement is reflected as an increase in the fidelity in the density matrix that represents our knowledge about the state. We can describe a purification protocol in terms of:

1. the number and type of input states;
2. the test procedure for certain propositions;
3. the scheduling algorithm used to select states for participation in purification.

In general, the required input states are two imperfect Bell pairs, with the goal being to produce one output Bell pair of higher fidelity.

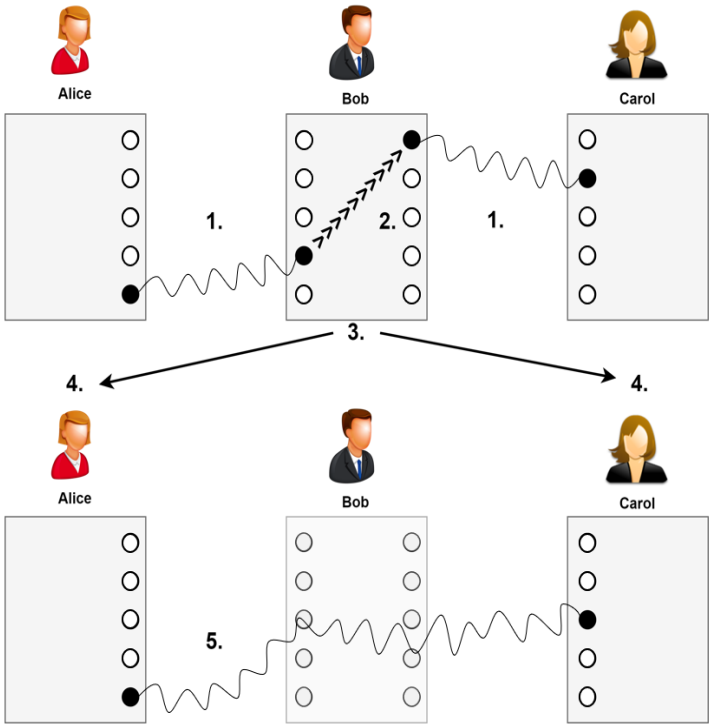



**Figure 2.8. Basic Entanglement Swapping**

To illustrate, we can consider the circuit for basic purification of Figure 2.9, used by Van Meter [Van Meter 2014]. Classical messages exchanged between Alice and Bob are indicated by the arrows.

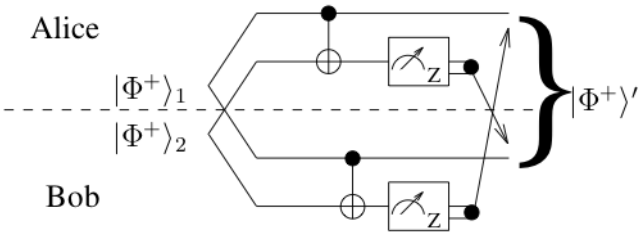



**Figure 2.9. Circuit for basic purification according [Van Meter 2014].**

When the Bell pairs, the gates, and measurements are all perfect, the two *CNOT* gates cancel, as illustrated in Equation (20) and we still have two unentangled $|\Phi^+\rangle$ pairs. Alice and Bob each have a 50% chance of finding 0 and 50% chance of finding 1 when the second pair is measured, and when they exchange their measurement results they will always find the same value.

$$|\Phi^+\rangle_1 |\Phi^+\rangle_2 \quad = \quad (|00\rangle + |11\rangle)(|00\rangle + |11\rangle) \tag{20}$$

$$
\begin{aligned}
&= |00\rangle|00\rangle + |00\rangle|11\rangle + |11\rangle|00\rangle + |11\rangle|11\rangle \\
&\xrightarrow{CNOT_A} |00\rangle|00\rangle + |00\rangle|11\rangle + |11\rangle|10\rangle + |11\rangle|01\rangle \\
&\xrightarrow{CNOT_B} |00\rangle|00\rangle + |00\rangle|11\rangle + |11\rangle|00\rangle + |11\rangle|11\rangle \\
&= |\Phi^+\rangle_1 |\Phi^+\rangle_2
\end{aligned}
$$

When a Bell pair suffer a bit flip error, $\rho = P|\Phi^+\rangle\langle\Phi^+| + (1-P)|\Psi^+\rangle\langle\Psi^+|$, we would find that is indeed a $|\Phi^+\rangle$ pair, with probability $P$, if we could test pair 1 directly, and we would find that it is a $|\Psi^+\rangle$ pair, with probability $1-P$. In the case when both pairs are in fact in $|\Phi^+\rangle$, the circuit operates as presented in Equation (20) and the probability it happens is $P^2$.

If either of the Bell pairs has a bit flip error, Alice and Bob will find different values when they measure their qubits and we have to discard pair 1, even though it might be good because we cannot tell if the error was in pair 1 or pair 2. If both Bell pairs have an error, Alice and Bob will find the same value. With probability $(1-P)^2$ , the error in pair 1 goes undetected due to the error in pair 2.

The probability of successful operation is $P^2 + (1-P)^2$, including the false positive case engendered by two errors, while the probability of operation failure is $2P(1-P)$. The resulting fidelity $(F_{ap})$ when operation succeed is

$$F_{ap} = \frac{P^2}{P^2 + (1-P)^2} \tag{21}$$

with the following final state

$$\rho_{ap} = F_{ap}|\Phi^+\rangle\langle\Phi^+| + (1-F_{ap})|\Psi^+\rangle\langle\Psi^+| \tag{22}$$

To check the quality of resulting fidelity is common to analyze the output fidelity as a function of input fidelity. Van Meter [Van Meter 2014] performed this analysis for basic purification of two identical Bell pairs with bit flip errors only, and perfect purification operations, as illustrated in Figure 2.10. $F_{ap} > F$ when $F > 0.5$ and for input fidelity greater than approximately 0.8, the improvement in fidelity is very good.

Various techniques for managing errors have been developed, some based on classical error correction and erasure correction techniques, others on uniquely quantum approaches [Devitt et al. 2013] [Terhal 2015]. Purification, in which two or more multiqubit states are manipulated to form one higher-fidelity state, uses few quantum memory resources and simple quantum operations, but operates only on well-understood states such as Bell states rather than arbitrary application data.

### 2.3.4. Quantum repeaters

A fundamental component of a quantum network is the quantum repeater. It allows the transportation of qubits over long distances, which is hindered by signal loss and decoherence inherent to most transport mediums such as optical fiber. Since absorption losses and depolarization error scale exponentially as distance increases, one cannot hope to cover any long distance between A and B in one leap [Pirandola et al. 2017a]. Repeaters appear

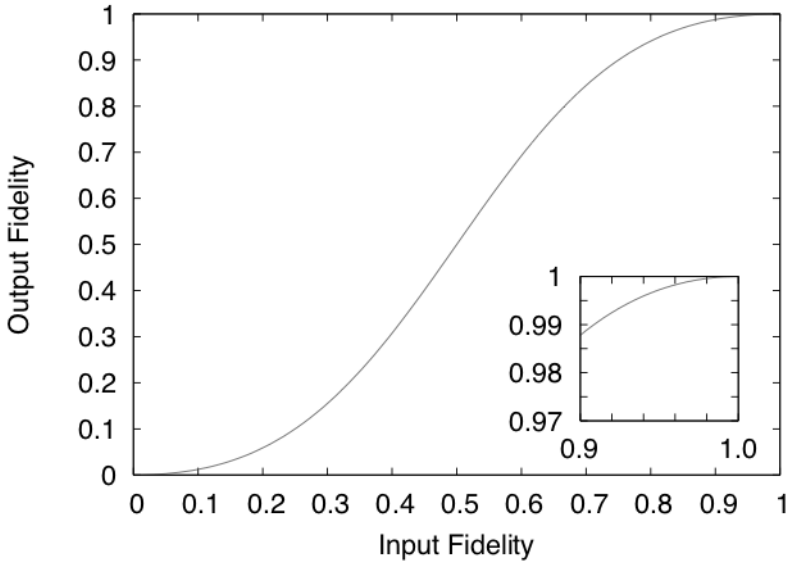



**Figure 2.10. Output fidelity as a function of input fidelity according [Van Meter 2014].**

in-between end nodes. Loss in telecommunications fiber is typically around 0.2 dB/km; high-quality fibers with loss of 0.17dB/km are available, and in laboratories, loss is as low as 0.12 dB/km. At 0.17 dB/km, the attenuation length is about 25 km. This value is commonly used in quantum repeater simulations, although NTT (Nippon Telegraph and Telephone) accomplished the remarkable feat of distributing time-bin entangled photons through 300 km of fiber [Inagaki et al. 2013].

In classical communication, amplifiers can be used to boost the signal during transmission, but in a quantum network amplifiers cannot be used since qubits cannot be copied - known as the no-cloning theorem. By necessity, a quantum repeater works in a fundamentally different way than a classical repeater. Quantum repeaters allow entanglement and can be established at distant nodes without physically sending an entangled qubit the entire distance [Bouwmeester et al. 1997]. A quantum repeater protocol executes three operations to create the long-range Bell state that can be used for quantum communication tasks such as QKD or teleportation. These operations are:

- Entanglement distribution: the process for creating entangled links between network nodes, as presented in Section 2.3.1.
- Entanglement purification: the process where we create a more highly entangled state from a number of lower quality ones, as described in Section 2.3.3.
- Entanglement swapping: the process in which a Bell-state measurement is performed within a node on two qubits which are halves of separate Bell states, as illustrated in Section 2.3.2. The Bell measurement allows us to generate a longer entangled link connecting adjacent repeater nodes.

The first operation is needed only between shorter-range adjacent nodes and thus

the success probability for generating the entangled link depends on the distance of the adjacent nodes, rather than the total communication distance. The second operation has the objective to not permit information present in the state has been lost. The third operation is the mechanism to extend the range of the entanglement.

While the quantum repeater protocol for generating long-range entanglement may seem quite straightforward in nature, its behavior is quite complex due to the various probabilistic elements inherent in the scheme [Sangouard et al. 2009]. Different generations of quantum repeaters have already been developed [Inagaki et al. 2013] [Munro et al. 2015] and progress has been significant in recent years both from an engineering perspective but also with new approaches [Kuzmin et al. 2019]. As the performance of these systems continues to improve they will also be able to take advantage of the developments in QKD, and secure quantum communication in general, in terms of their integration in standard fiber-optic networks.

Early architectures of repeaters, in essence, used teleportation to extend entanglement, and purification to detect errors introduced in the process [Munro et al. 2013]. The process of entanglement swapping uses teleportation to splice two Bell pairs spanning adjacent short distances into one pair over the corresponding longer distance, as presented in Figure 2.8. Entanglement swapping is independent of the distances between A and B, and between B and C. Only local quantum operations are required, supported by classical communication. Purification is used to compensate for the errors introduced, as described in Section 2.3.3. Local quantum operations are performed at both nodes on two Bell pairs, then one of the Bell pairs is measured. The measurement results are exchanged and compared. If they agree, the pair's fidelity has improved, and it is kept for reuse. If the measurement results disagree, the pair is discarded.

From these concepts to execute a distributed algorithm that creates entangled quantum states between nodes that are far apart, different researchers [Jiang et al. 2009] [METER et al. 2011] presented simple protocol stacks for networks of quantum repeaters that considers all the necessary classical messages and which can be easily adapted for different approaches, as illustrated in Figure 2.11.

The physical entanglement layer represents the physical interaction that creates Bell pairs between two different stations. Many technologies for this layer are under development [Munro et al. 2013]. The second layer, Entanglement Control, is responsible for managing the single-hop physical entanglement process, selecting qubits to attempt entanglement at each end of the link, and utilizing classical messages to report the results. The third layer of the protocol, Error Management, is responsible for choosing two Bell pairs, and electing one pair to have its fidelity boosted and the other to be sacrificed, assuring that both stations make the same decisions. The fourth layer, Quantum State Propagation, is responsible for administering the Bell pairs, especially for networks with shared resources. Important decisions, such as whether to purify or swap first or when to swap, need to be carefully taken. The Application layer will determine if end-to-end entanglement is required, or if our quantum states can be measured on a pay-as-you-go basis. Currently, the most important existing application is QKD (Quantum Key Distribution). However, the application may be a sensor network, or a numeric computation or decision algorithm based on shared state, as we will see in Section 2.3.5.

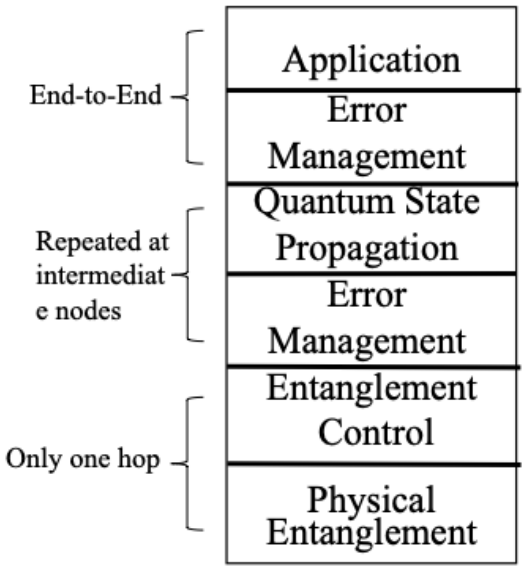



**Figure 2.11. Basic protocol stack architecture for networks of quantum repeaters.**

It is important to mention that QKD applications can be performed not only creating an end-to-end quantum channel but also using trusted repeaters. Consider two end nodes Alice(A) and Bob(B), and a trusted repeater R in the middle. A and R now perform quantum key distribution to generate a key $K_{AR}$. Similarly, R and B run quantum key distribution to generate a key $K_{RB}$. A and B can now obtain a key $K_{AB}$ between themselves as follows: A sends $K_{AB}$ to R encrypted with the key $K_{AR}$. R decrypts to obtain $K_{AB}$. R then re-encrypts $K_{AB}$ using the $K_{RB}$ and sends it to B. B decrypts to obtain $K_{AB}$. A and B now share the key $K_{AB}$. The key is secure from an outside eavesdropper, but clearly the repeater R also knows $K_{AB}$. This means that any subsequent communication between A and B does not provide end to end security, but is only secure as long as A and B trust the repeater R.

A true quantum repeater allows the end to end generation of quantum entanglement, and thus - by using quantum teleportation - the end to end transmission of qubits. In quantum key distribution protocols, one can test for such entanglement. This means that when making encryption keys, the sender and receiver are secure even if they do not trust the quantum repeater. Any other application of a quantum Internet also requires the end to end transmission of qubits, and thus a quantum repeater.

To make maximum use of communication infrastructure, we also require optical switches capable of delivering qubits to the intended quantum processor. These switches need to preserve quantum coherence, which make them more challenging to realize than standard optical switches. Current commercial switches have various problems that make them unsuitable for rerouting entangled photons. Those that are made of micro-electromechanical components keep entangled states intact but operate too slowly. Other optoelectronic switches either add too much noise so that single photons are difficult to detect, or they completely destroy the quantum information. The utilization of a quantum switch provides significant advantages for a number of problems, ranging from

quantum computation and quantum information processing, through non-local games to quantum communication [Caleffi 2020].

### 2.3.5. Quantum Key Distribution

In literature, there are several types of quantum applications. They are often classified into two categories, distributed agreement protocols, and distributed computation, although the underlying theory is essentially the same [Van Meter 2014]. However, in this section, our focus will be on Quantum Key Distribution (QKD), the most practical, commercially attractive use of quantum networks in the near term. We will present how the protocol works and the network requirements to support application.

The main objective in QKD is the use of quantum mechanics to detect the presence or absence of an eavesdropper. QKD systems generate shared, secret random numbers between two distant parties. Shared random numbers, if provably secret, can be used as cryptographic keys, allowing secure communication across physically insecure networks such as the Internet. An important and unique property of QKD is the ability of the two communicating users to detect the presence of any third party trying to gain knowledge of the key. This results from a fundamental property of quantum mechanics: the process of measuring a quantum system, in general, disturbs the system. A third party trying to eavesdrop on the key must in some way measure it, thus introducing detectable anomalies. By using quantum superpositions or quantum entanglement and transmitting information in quantum states, a communication system can be implemented that detects eavesdropping. If the level of eavesdropping lies below a certain threshold, a key can be produced that is guaranteed to be secure (i.e., the eavesdropper has no information about it), otherwise no secure key is possible and key generation is aborted.

Artur Ekert proposed, in 1991, a QKD protocol currently called E91 [Ekert 1991a], using entangled pairs of photons. The entangled states are perfectly correlated in the sense that if Alice and Bob both measure whether their particles have vertical or horizontal polarizations, they always get the same answer with 100% probability. The same is true if they both measure any other pair of complementary (orthogonal) polarizations. This necessitates that the two distant parties have exact directionality synchronization. However, the particular results are completely random; it is impossible for Alice to predict if she (and thus Bob) will get vertical polarization or horizontal polarization. Any attempt at eavesdropping by Eve destroys these correlations in a way that Alice and Bob can detect. The correct operation of E91 does require a functioning quantum network capable of generating the required Bell pairs. The fidelity is also important, because, with low fidelity, the eavesdropper detection becomes more difficult and consumes a larger fraction of the end-to-end Bell pairs.

QKD can be incorporated into a production classical network in different ways [Pirker and Dur 2019]. A simple arrangement is to securely connect two networks in two far locations, that can belong to the same organization or not. One approach is to use a virtual private network (VPN) to connect the two locations [Van Meter 2014]. Implementations of QKD are well beyond the experimental phase [Dodson et al. 2009]. A few commercial products are available, and metropolitan-area testbed networks exist in Boston, Vienna, Geneva, Barcelona, Durban, Tokyo, several sites in China and elsewhere

throughout the world. In fact, the BB84 [Bennett and Brassard 1984] technique deployed in most links in these networks does not use entangled quantum states, although another approach, developed by Artur Ekert, does [Ekert 1991b]. QKD has also been integrated into custom encryption suites and the Internet standard IPsec suite and has been proposed for use with the TLS protocol commonly used on the World Wide Web [Mink et al. 2010].

QKD can also be viewed as a form of sensor network: the goal of the underlying quantum operation is the same, physical detection of eavesdropping on the quantum channel [Van Meter 2014]. Distributed entanglement is an extremely sensitive physical state, and can be used as a physical probe for other applications, like improving the resolution of optical telescopes using interferometry [Gottesman et al. 2012] and comparing the relative time of two clocks separated by a distance [Kómár et al. 2014]. Both of these applications are far from practical given both the current state of the technology and the very demanding nature of the existing proposals, but they serve as important signposts on the road to the merger of quantum information and real-world sensors and actuators.

### 2.3.6. Quantum networks and Quantum Internet

We presented in the previous sections that quantum networks form an important element of quantum computing and quantum communication systems. Quantum networks facilitate the transmission of information in the form of qubits between physically separated end nodes. Quantum networks, like classical networks, will involve nodes and links and a layered communication architecture with individual protocol modules communicating vertically up and down a protocol stack and horizontally with peers. There are, however, fundamental differences that make the merger of classical and quantum networking concepts less than straightforward [Di Franco and Ballester 2012].

In principle, we can consider two main approaches to construct quantum networks [Dahlberg et al. 2019]. On the first alternative, quantum networks could simply forward quantum information directly, which however needs to be protected against noise and decoherence using quantum error correcting codes, and repeatedly refreshed at intermediate stations where error correction is performed. On the second one, quantum networks may use entanglement, a property, as presented in Section 2.2.3, that is only accessible in quantum physics. Constructing quantum networks by using entanglement has one significant advantage compared to the first approach: the entanglement topology of a network, which determines in that case also the boundaries and ultimately the structure of a network, is completely independent of the underlying physical channel configuration.

As shown in Section 2.3.4, a crucial element to establishing long-distance entanglement are quantum repeaters, and multiple proposals for repeater-based networks have been put forward [METER et al. 2011] [Rubino et al. 2017]. Although most schemes are based on bipartite entanglement, where Bell pairs are generated between nodes of the network, future quantum networks shall not be limited to the generation of Bell-pairs only, because many interesting applications require multipartite entangled quantum states [Pirker and Dur 2019].

The basic structure of a quantum network and more generally a quantum Internet is analogous to a classical network [Caleffi and Bianchi 2018]. Besides quantum repeaters, we have end nodes, quantum channels, and quantum switches. Applications run

in end nodes. These end nodes can be quantum processors containing at least one qubit. Most applications of a quantum Internet require only very modest quantum processors. For most quantum Internet protocols, such as QKD, it is sufficient if these processors are capable of preparing and measuring only a single qubit at a time. On the other hand, some applications of a quantum Internet require quantum processors of several qubits as well as a quantum memory at the end nodes. Quantum network nodes can exchange classical control information over standard classical communication channels. This may be by means of a direct physical connection or via, for example, the Internet.

Layering is a natural means of dividing functionality, and the associated modularity allows us to replace individual functions more or less independently. In this context, it is essential to develop methods that allow quantum protocols to connect to the underlying hardware implementation transparently and to make fast and reactive decisions for generating entanglement in the network in order to mitigate limited qubit lifetimes. However, only preliminary functional allocation of a quantum network stack has been proposed, and just first versions of physical and link layer protocols have been developed [Dahlberg et al. 2019] [Pirker and Dur 2019] [Van Meter 2014].

Stephanie Wehner et. al [S. Wehner and Hanson 2018] propose stages of development toward a full-blown quantum Internet. They suggested stages that are functionality driven: Central to their definition is not the difficulty of experimentally achieving them but rather the essential question of what level of complexity is needed to actually enable useful applications. Each stage is interesting in its own right and distinguished by a specific quantum functionality that is sufficient to support a certain class of protocols, as illustrated in Figure 2.12.

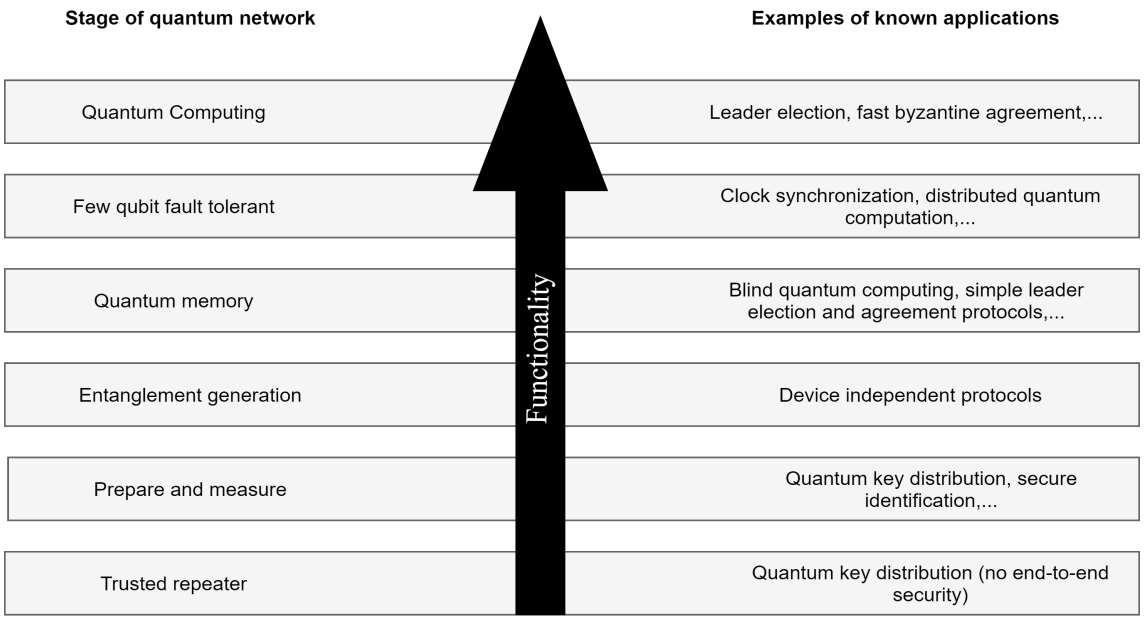



**Figure 2.12. Stages of Quantum Internet according Stephanie Wehner et al. [S. Wehner and Hanson 2018].**

Each stage is characterized by an increase in functionality at the expense of greater technological difficulty. A specific implementation of a quantum Internet may, like for a classical network, be optimized for distance, functionality, or both. The objective of a network is to provide any end nodes (connected to the network) with the means to exchange data, making three end nodes the smallest instance of a true network.

We will briefly describe only the first two stages because they are ones that have been realized in practice in some way. A trusted repeater network, first stage, has at least two end nodes and a sequence of short distance links that connect nearby intermediary repeater nodes. Each pair of adjacent nodes uses QKD to exchange encryption keys. The main characteristic of trusted repeater networks is that they do not allow the end-to-end transmission of qubits. Currently, outside the laboratory, only trusted repeater networks have been realized in metropolitan areas [Wang et al. 2014b].

The second stage, prepare and measure networks, enables end-to-end QKD without the need to trust intermediary repeater nodes and already allows other protocols. It allows any node to prepare a one-qubit state and transmit the resulting state to any other node, which then measures it. This stage is the first to offer end-to-end quantum functionality and demands the use of quantum repeaters to bridge long distances via intermediate qubit storage or error correction, as well as routers to forward the quantum state to the desired node. Several recent experiments have demonstrated elements belonging to this stage [Pirker and Dur 2019]

In 2011, Van Meter et. al [METER et al. 2011] introduced the concept of a Quantum Recursive Network Architecture (QRNA), developed from the emerging classical concept of recursive networks, extending recursive mechanisms from a focus on data forwarding to a more general distributed computing request framework. Recursion abstracts independent transit networks, as single relay nodes, unifies software layering and virtualizes the addresses of resources to improve information hiding and resource management. The architecture is useful for building arbitrary distributed states, including fundamental distributed states such as Bell pairs and GHZ, W and cluster states.

Routing is another interesting aspect of quantum networks [Pant et al. 2019a]. Routing messages to the right destination in a network has seen enormous attention in the classical literature, and the term routing is also used for a number of different concepts in quantum networks. It is highly likely that concepts from both domains will form an important ingredient in designing quantum networks [Chakraborty et al. 2019].

Applying the appropriate simplifications, routing in a quantum network can be understood as routing on virtual quantum links (VQLs). However, such VQLs are rather unusual from a classical perspective in that they can be used only once, and require one qubit of quantum memory at each endpoint to be maintained [Pant et al. 2019b]. While the requirements of a VQL for one qubit of memory at each node appears benign from a classical perspective, it is rather significant in a quantum network. First of all, due to current technological limitations, each network node can store only very few qubits. What's more, such quantum storage is typically rather noisy, meaning that each qubit has a limited lifetime. The latter can, in theory, be overcome by performing error-correction at each network node at the expense of using additional qubits. In resume, the VQLs can be assigned deliberately and dynamically, carry a certain cost, can only be used once, and

which may expire after a given time.

As a result, communication in a quantum network can, in principle, be understood entirely as transformations performed on the graph of VQLs: Given existing VQLs, to send a qubit from two nodes A and B we consume VQLs to create a new entangled link - a new VQL - directly between A and B, followed by teleportation of the qubit over said VQL [Schoute et al. 2016].

## 2.4. Current Scenario and Research Challenges

The Quantum Internet is envisioned as the final stage of the quantum revolution, opening fundamentally new communications and computing capabilities, including distributed quantum computing. This section presents the current status of the quantum Internet, along with challenges, and research opportunities in this emerging area. The focus is on laying the groundwork to adapt Internet design principles to the development of quantum networks. We discuss the key research challenges and open problems related to the design of a quantum network, which harnesses quantum phenomena, such as entanglement and superposition, to share quantum states among remote quantum devices.

### 2.4.1. Current scenario

The quantum Internet describes a collection of distributed quantum nodes, separated by a range of distances over which one desires to perform some quantum communication protocol that can support, for example, distributed quantum computation or distributed sensing [Pirandola and Braunstein 2016]. There are currently numerous quantum communication and cryptographic protocols identified, including security distribution for encryption [Mink et al. 2010, Kumar et al. 2019], quantum-certified random number generation in the form of random number beacons and personal devices, secret-sharing [Hillery et al. 1999, Williams et al. 2019], quantum fingerprinting [Guan et al. 2016] and other multi-party computation protocols, such as secure quantum voting, byzantine agreements, and multi-party private auctions [Broadbent and Schaffner 2015].

As we mentioned in Section 2.3.6, the current experimental status of long-distance quantum networks is at the lowest stage with several commercial systems for QKD on the market. The first extended trusted repeater networks have already been implemented over metropolitan distances [Wang et al. 2014a], and a long-distance implementation has recently been completed [Kumar et al. 2019]. The hardware required for the lowest stage has been described in detail in [Diamanti et al. 2016]. Realizing the next stages with end-to-end quantum functionality over long distances demands, basically, the use of quantum repeaters to bridge long distances via intermediate qubit storage or error correction, as well as routers to forward the quantum state to the desired node. Several recent experiments have demonstrated elements belonging to this and higher stages at short distances, suggesting that higher-functionality networks are within reach [Cacciapuoti et al. 2020]. To put these advances into the right perspective, we briefly summarize the main requirements for the essential quantum Internet hardware.

Quantum links between the repeater stations and the end nodes are established via photonic channels. Two types of photonic channels can be considered: free-space channels, potentially via satellites [Yin et al. 2017], and fiber-based channels. Each has its

own advantages and disadvantages, and a future quantum Internet may use a combination of them, similar to the current classical Internet. Hybrid architectures will probably be used for connections faring both the use of cryo-cables (expensive and necessarily limited in length) and of optical fibers or free space photonic links. We require these channels to exhibit minimal photon loss and decoherence.

The end nodes need to meet the following requirements for the quantum Internet to reach its full potential:

- Robust storage of quantum states during the time needed to establish entanglement between end nodes. This robustness must persist under quantum operations performed on the end node.
- High-fidelity processing of quantum information within the node. For more advanced tasks, multiple qubits will be required, making the end nodes similar to small-scale quantum computers.
- Compatibility with photonic communication hardware: efficient interface to light at the relevant wavelength.

Several experimental platforms are currently being pursued for the end nodes. Each of these combines well-controlled matter-based qubits with a quantum optical interface via internal electronic transitions.

For quantum repeaters, the requirements are less strict than for the end nodes. Depending on the architecture of the repeaters, the storage of quantum states may only be required for the time needed to establish entanglement between the nearest active nodes, substantially different from the storage time required for the end nodes. Also, the qubit processing capabilities required are limited, and therefore systems different from the end nodes can be considered.

As mentioned in Section 2.3.4, the quantum switch is another essential component for a quantum Internet. Multiple physical implementations of the quantum switch have been proposed and experimentally realized with photons, with the control qubit represented by polarization or orbital angular momentum degrees of freedoms. Preliminary results of the quantum switches to face with the noise degradation introduced by the entanglement distribution are very good. However, a substantial amount of conceptual and experimental work has to be developed in order to tackle the challenges and open problems associated with the utilization of the quantum switch in the Quantum Internet [Rubino et al. 2017] [Awschalom et al. 2019].

At the present time, however, quantum networking in the real world consists of three research programs and commercialization efforts: the first one is Quantum Key Distribution (QKD) that adds unbreakable coding of key distribution to public-key encryption, as we presented in Section 2.3.5.

The second one is cloud/network access to quantum computers. It is core to the business strategies of leading quantum computer companies. Cloud-based quantum computing is the invocation of quantum emulators, simulators or processors through the cloud. Increasingly, cloud services are being looked on as a method for providing access to

quantum processing. IBM already had connected a small quantum computer to the cloud and it allows users to execute simple programs on the cloud [IBM 2020]. Many people from academic researchers and professors to schoolkids have built programs that run many different quantum algorithms using the program tools. Some consumers hope to use fast computing to model financial markets or to build more advanced AI systems [Chen et al. 2018].

The last one is quantum sensor networks, which could exploit the correlations across an array of sensors, linking them to each other with quantum mechanical means [Degen et al. 2017]. Quantum sensors exploit superposition, entanglement, squeezing, and backaction evasion to make measurements with a precision better than the Standard Quantum Limit (SQL), with the ultimate goal of reaching the Heisenberg Limit. Sensor networks improve the sensitivity and scalability of the resulting entangled system simultaneously allowing it to benefit from the long-distance baseline between the sensors. Some promising options are quantum networks of atomic clocks, phase-sensitive quantum networks, quantum networks of magnetometers [Proctor et al. 2017].

For the past 15 years, major service providers and research institutions worldwide have run quantum network trials. We are now entering a period in which permanent quantum networks are being built. These are designed initially to support quantum encryption services, but will soon also provide the infrastructure for quantum computing. As quantum networks are deployed, they will eventually create opportunities at the service level, but more immediately at the components and modules level. This is because quantum networks will require a slew of new optical networking technologies to make them function effectively. The market for quantum networking is projected to reach $5.5 billion by 2025, according to a new report from Inside Quantum Technology (IQT) [IQT 2020].

### 2.4.2. Research opportunities

In this subsection, we discuss the key research challenges and open problems related to the design of a quantum network, which harnesses quantum phenomena with no-counterpart in the classical reality, such as entanglement and teleportation, to share quantum states among remote quantum devices.

Exploring how to build the quantum Internet, a vast network of quantum computers and other quantum devices, will catalyze new technologies that accelerate today's Internet, improve the security of our communications, and allow dramatic advances in computing [Caleffi 2020]. The difficulty of every item in the design of a network grows as the scale of the network increases. This is true for both, classical and quantum networks. The main challenges in scaling networks to Internet-scale and beyond are: heterogeneity, especially of deployed technologies and local conditions; sheer scale, affecting routing and naming in particular; dealing with out of date information about current network conditions (e.g. routing or congestion) and the success or failure of requested operations; meeting the needs of participating organizations, such as privacy, desired traffic transit policies and autonomous management; and dealing with misbehaving nodes on the network, whether the misbehavior is deliberate or accidental [Cacciapuoti et al. 2020].

As we presented in Section 2.2.2 qubits are very fragile: any interaction of a qubit with the environment causes decoherence, i.e., a loss of information from the qubit to

the environment as time passes, and isolation is hard to achieve in practice given the current state-of-the-art of quantum technologies. Furthermore, perfect isolation is not desirable, since computation and communication require interaction with the qubits, e.g., for reading/writing operations. Although a gradual decrease of the decoherence times is expected with the progress of the quantum technologies, the design of a quantum network must carefully account for the constraints imposed by quantum decoherence.

Decoherence is not the only source of errors. Errors practically arise with any operation on a quantum state due to imperfections and random fluctuations. Here, a fundamental figure of merit is the quantum fidelity, presented in Section 2.2.6. From a communication engineering perspective, the joint modeling of errors induced by the quantum operations, together with those induced by entanglement generation/distribution, is still an open problem.

Furthermore, the no-cloning theorem prevents the adoption in quantum networks of classical error recovery techniques, depending on information cloning, to preserve quantum information against decoherence and imperfect operations. Recently, many quantum error correction techniques have been proposed as in [Chandra et al. 2018]. However, further research is needed. In fact, quantum error correction techniques must handle not only bit-flip errors, but also phase-flip errors, as well as simultaneous bit- and phase-flip errors. This differs from classical networks that only have to consider the bit-flip error.

One fundamental difference with respect to classical networks, where broadcast is widely exploited for implementing several link layer and network layer functionalities, such as medium access control and route discovery is the impossibility of transmitting quantum information to more than a single destination due to no-broadcasting theorem [Barnum et al. 2007], a corollary of the no-cloning theorem. As a consequence, the link layer must be carefully re-thought and re-designed, and effective multiplexing techniques for quantum networks should be designed to allow multiple quantum devices to be connected to a single quantum channel (e.g. a fiber). Access to the medium could be based for example on photon-frequency-division for the entanglement distribution.

Entanglement distribution determines the connectivity of a quantum network in term of capability to perform teleporting among quantum devices. Hence, novel quantum routing metrics are needed to ensure effective entanglement-aware path selection. Furthermore, the teleportation process destroys entanglement as a consequence of the BSM at the source. Hence, if additional qubits need to be teleported, new entangled pairs need to be created and distributed between the source and the destination. This constraint has no-counterpart in classical networks and it must be carefully accounted for in an effective design of the network layer [Pant et al. 2019b, Kumar et al. 2019].

In relation to the deployment of a quantum Internet, there are several challenges for the near future. At first, quantum computers will be available in few, highly specialized, data centers capable of providing the challenging equipment needed for quantum computers. Companies and users will be able to access quantum computing power as a service via cloud. In this regard, the quantum cloud market is estimated nearly half of the whole 10 billion dollar quantum computing market by 2024 [Market 2018]. IBM already allows researchers to design and execute quantum algorithms through classical

cloud access to isolated 5-, 16- and 20-qubits quantum devices.

To extend the range of fiber-based entanglement distribution beyond a few hundred kilometers, quantum repeaters are required. One important benchmark for quantum repeaters is the repeater-less bound, which imposes the fundamental limit of the direct quantum communication protocols. Recently, there have been significant advances in experimentally demonstrating key elements of a quantum repeaters in an integrated system. An important recent highlight is the experimental demonstration of memory-enhanced quantum communication surpassing repeater-less bound in a proof-of-concept laboratory setting [Bhaskar et al. 2020] [Pirandola et al. 2017b]. This paves the way towards the demonstration of a full quantum repeater, which in turn will enable scalable large-scale quantum networks.

An important challenge in extending the point-to-point entangled links into true networks is the problem of efficient storage of quantum states [Hucul et al. 2014]. Ideally, it is necessary to have a quantum memory that is capable of storing and releasing quantum states on the level of individual qubits and on-demand. Storage and on-demand retrieval have already been achieved [Delteil et al. 2017], although efficiencies are still to be improved.

Another challenge is that most of the above systems do not intrinsically couple to light in the telecom band. To fulfill the compatibility requirement with photonic communication hardware, wavelength conversion at the single-photon level can be used [Zaske et al. 2012]. While existing quantum interfaces between modules have seen dramatic improvements, most systems still have not reached the regime where connection between the modules can be utilized for reliable transfer of qubits within the timescale required for distributed quantum computation. The first realizations of a Quantum Internet probably will be small clusters of quantum processors within a data center. Architectures will have to take into account the high cost of data buses (economically and in terms of quantum fidelity) limiting both the size of the clusters and the use of connections for processing.

Seamless integration of the communication interfaces with the computational functions of the modules can also introduce some challenges. For example, in heralded entanglement generation protocols, the qubit-photon entanglement generation protocols can lead to decoherence of nearby qubits storing information. For these systems, novel integration approaches must be developed so that communication and local data processing can co-exist. For solid-state qubits (such as superconducting qubits) that use photons in the microwave range of the electromagnetic spectrum, communication over room-temperature channels becomes impractical. Given the fragile nature of quantum entanglement and the challenges posed by the sharing of quantum resources, a substantial amount of research is needed in the development both of novel networking protocols and of quantum and classical algorithms [Kuzmin et al. 2019].

The ability to control, optimize, and recover from failures are critical in the design and operation of large-scale networks [Awschalom et al. 2019]. Achieving these capabilities is a challenge that typically requires networks to carry network management and control information in addition to user data. There are two distinct methods to implement this: in-band control, if control and management information is carried on the same chan-

nel as user data, and out-of-band, if it is carried on a separate channel.Clearly, quantum networks need out-of-band control and signaling, since any attempt to read and process control information carried in the quantum channel will destroy its content.

Modeling and performance analysis are important, both in the design phase to evaluate and compare the merits of a variety of quantum network protocols and architectures, as well as for real-time performance analysis and troubleshooting after the network is built [Vardoyan et al. 2019b, Vardoyan et al. 2019a]. Simulations will be needed to study network properties including quantum state and entanglement throughput, latency, scalability, reliability, and availability. One interesting alternative is NetSquid [Elkouss et al. 2020], capable of simulating the decay of quantum information over time together with noisy operations and stochastic feedback loops. Since generic quantum systems consisting of even hundreds of qubits cannot be fully simulated on classical computers, ways to effectively employ reduced models are required. Some methods already exist that will clearly be useful, such as Monte Carlo simulations of systems whose operations only include Clifford gates. Some questions, for example, the extent to which various protocols are able to avoid bottlenecks, will be able to be addressed by purely classical simulations, using methods similar to those already developed by the classical networking community. Other questions, pertaining to the physical layer, will require simulation of the dynamics of optical channels and their interaction with the systems that comprise sending and receiving circuits. Such simulations will likely require the use of much more sophisticated methods such as matrix-product-state methods or tensor-network methods.

Quantum networks are complex, challenging engineered systems that require sophisticated solutions for their operations and control, with many of those solutions yet to be developed [Caleffi 2020]. Indeed, many of the control plane technologies in use in modern classical networks are not suitable for the quantum data plane that cannot be subjected to O-E-O conversion, as discussed above. Quantum network management and operation will be particularly challenging due to the quantum nature embedded in the control plane and/or the data plane. The task is further complicated by the need for quantum networks to co-exist with conventional networks. In addition, monitoring of quantum networks requires measurements of complex conventional and quantum signals, along with inferences and analytics to distill knowledge and make control decisions [Ndousse-Fetter et al. 2019].

There are unique and extremely important security questions involved in the reliable, trustworthy operation and control of quantum resources to support quantum computation and quantum sensing efforts [Cacciapuoti et al. 2020]. Within the framework of coexistence infrastructures, security vulnerabilities of conventional networks carry over, and those of (newer) quantum components need to be explored and addressed. Furthermore, novel crossover vulnerabilities may potentially exist, wherein one modality may be exploited to compromise the other. Indeed, these aspects must be addressed from the start as an integral part of the design and analysis of quantum networks.

Another interesting research opportunity is the use of software defined network (SDN) technologies [McKeown 2009] [Aguado et al. 2019]. In particular, the success of QKD networks radically depends on the degree to which they can be adopted in the existing infrastructure. The integration of SDN, through the development of standard pro-

tocols and interfaces, has allowed new services and systems to be seamlessly integrated in telecommunications networks. The flexibility brought by SDN reduces drastically the effort of integrating new devices and technologies in the network and allows address the design of versatile quantum networks through the development of programmable quantum switches [Humble et al. 2018].

Finally, quantum teleportation requires the integration of classical and quantum communication resources. Classical communication resources will be likely provided by integrating classical networks such as the current Internet with the Quantum Internet. However, Currently, there is no notion of a "Quantum Packet," - a photonic quantum state along with appropriate headers that function as a single data unit that traverses the quantum network. As no such network stack presently exists for a quantum Internet, this represents a completely unexplored open problem, and its solution requires a multidisciplinary effort, spanning the breath from communications theory and engineering communities to the networking engineering one [S. Wehner and Hanson 2018].

## 2.5. Concluding Remarks

The potential to completely change markets and industries - such as commerce, intelligence, military affairs [Cacciapuoti et al. 2020], tackling classes of problems that choke conventional machines, such as molecular and chemical reaction simulations, optimization in manufacturing and supply chains, financial modeling, machine learning, and enhanced security, has led tech giants, like IBM, Google, Intel, Alibaba, and others to a race for building quantum computers. The Quantum Internet has been proposed as the key strategy to significantly scale up the number of qubits for long-distance communication of quantum and classical information. However, quantum computing and networking technologies are still at an early stage of research and development (R&D). This section presents the general conclusions on the topic addressed in the text, as well as a summary of the main contributions of the chapter.

Quantum communication networks are a nascent technology. As a society, we have come to expect more out of computing, especially networking, and cannot imagine a day without it. We want networks that are compatible with our computing needs, faster, secure, and accessible anywhere and anytime. The emergence of a quantum-computing paradigm that is radically different and incompatible with the current model of communications presents a unique challenge. Quantum networks have a solid and well-documented quantum-mechanical theoretical foundation but not much is known about translating it into practical implementation for most of the science applications that we presented in this chapter.

As we presented in Section 2.3, from a communication engineering perspective, the design of the Quantum Internet is not an easy task at all. In fact, it is governed by the laws of quantum mechanics, thus phenomena with no counterpart in classical networks - such as no-cloning, quantum measurement, entanglement and teleporting - would impose terrific constraints on the network design. For instance, classical network functionalities, such as error-control mechanisms (e.g., ARQ) or overhead-control strategies (e.g., caching), are based on the assumption that classical information can be safely read and copied. But this assumption does not hold in a quantum network. As a consequence, the

design of a quantum network requires a major paradigm shift to harness the key peculiarities of quantum information transmission, i.e., entanglement and teleportation.

As the size of quantum systems grows, in terms of number of qubits in the case of quantum computers, or physical size/spatial separation in the case of quantum networks, so do the challenges related to connecting different parts of the system while maintaining quantum entanglement across it. For example, long-range communication networks rely on establishing, distributing and maintaining entanglement across thousands of kilometers. This is challenging due to unavoidable signal losses in the communication channels. As we mentioned in Section 2.3.4, depending on the tools used for suppressing the imperfections, the quantum information community has identified the following three generations of quantum repeaters: The first generation uses heralded entanglement generation and heralded entanglement purification, which can tolerate more errors but requires two-way classical signaling over the entire chain of quantum repeaters; such signaling then implies that the requisite quantum memory lifetimes/coherence times must be substantially longer than the round-trip communication times. The second generation introduces quantum encoding and classical error correction to replace the entanglement purification with classical error correction, handling all operational errors, which is more demanding in physical resources but requires only two-way classical signaling between neighboring repeater stations, and consequently further improves the quantum communication rate. The third generation of quantum repeaters would use quantum encoding to deterministically correct both photon losses and operation errors. By entirely eliminating two-way classical signaling, the third generation of QRs would promise extremely high entanglement distribution rates that can be close to classical communication rates, limited only by the speed of local operations, in turn, limited by, e.g., photon source rates, detector saturation rates, and timing jitter, etc.

Another key aspect of a fully functioning quantum Internet is the potential for unconditional information security - a feature of using quantum information that is not possible with classical information processing. A further benefit of using quantum secured information will be that the lifetime of the security is "infinite"; it will be secure against any advances in computation capability that may occur in the future. There have been many cryptographic tasks in which quantum-secured versions have been conceived. For all of these tasks, quantum interconnects are required because of the need to preserve entangled quantum states.

As we mentioned in this chapter, to realize fully the potential of a quantum internet, significant convergent work is still needed to improve the physical hardware. Theoretical work is also required to develop efficient information processing techniques to preserve quantum information and determine the most robust and secure network connectivity. The development of quantum-secured devices and protocols could transform the cryptographic landscape.

Quantum networks, like many other innovations, which originate from basic research in academia and national labs, face technology transfer challenges despite their overwhelming potential to increase the nation's capabilities and benefit its society. Even though HPC and high-performance optical networks have been developed in concert to improve computing capability, major information technology providers investing in quan-

tum computing, such as Google, IBM, and Microsoft, and others in the telecommunications sector still view quantum networks as a high-risk effort. This means that the government must play an active role in prioritizing and matching investments from the private sector. As a neutral actor, the government could also facilitate the development of standards that will be critical in building inter-operable subsystems critical for quantum telecommunications.

Attentive to this issue, the main countries in the world have been defining as strategic vision focuses on R&D efforts to advance the development of foundations for the quantum Internet [U. S. Quantum 2020, Canada 2020, European Alliance 2020]. USA's strategy, for example, was developed through the National Quantum Initiative Act (NQIA), the National Quantum Coordination Office (NQCO) and the National Science and Technology Council's Subcommittee on Quantum Information Science (SCQIS) and reflects deep community input from SCQIS request for information responses of 2018-2019 and from recent workshops hosted by Federal agencies [Awschalom et al. 2019]. Quantum Internet Alliance aims to develop a Blueprint for a pan-European entanglement-based Quantum Internet, by developing, integrating and demonstrating all the functional hardware and software subsystems. China already has completed its own quantum key network using satellite communication. The first one, known as Micius, in English, was launched in 2016 [Liao et al. 2018].

Lastly, quantum networking is a nascent interdisciplinary field in quantum information processing. It is drawing interest from disparate fields such as quantum physics, telecommunications engineering, optical communications, computer science, cybersecurity, and domain science that have not traditionally worked together. Such collaboration is needed to solve a problem as complex as developing a general-purpose quantum network. However, these communities do not currently have a shared vocabulary or world-view, and many do not understand the specifics clients' requirements for interconnecting quantum computers and/or quantum sensors. Therefore, this effort will require a relatively long period of collaboration between researchers from these various communities, so that they can achieve a shared understanding of the problems and of the potential solutions. Once this shared understanding is achieved, it will be possible to perform a more detailed analysis of alternative concepts to determine whiches quantum network architectures will better satisfy the clients' needs.

## Acknowledgments

This research was supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and by the RNP-NSF joint call for research and development in cybersecurity, trough INSaNE (Improving Network Security at the Network Edge) project, funded by the National Science Foundation and the Brazilian Ministry of Science, Technology, Innovation, and Communication (MCTIC) through RNP and CTIC. This work was supported in part by the National Science Foundation under grant CNS-1617437.

## References

[Abruzzo et al. 2013] Abruzzo, S., Bratzik, S., Bernardes, N. K., Kampermann, H., van Loock, P., and Bruß, D. (2013). Quantum repeaters and quantum key distribution: Analysis of secret-key rates. *Physical Review A*, 87(5):052315.

[Aguado et al. 2019] Aguado, A., Lopez, V., Lopez, D., Peev, M., Poppe, A., Pastor, A., Folgueira, J., and Martin, V. (2019). The engineering of software-defined quantum key distribution networks. *IEEE Communications Magazine*, 57(7):20–26.

[Awschalom et al. 2019] Awschalom, D., Berggren, K. K., Bernien, H., Bhave, S., Carr, L. D., Davids, P., Economou, S. E., Englund, D., Faraon, A., Fejer, M., Guha, S., Gustafsson, M. V., Hu, E., Jiang, L., Kim, J., Korzh, B., Kumar, P., Kwiat, P. G., Loncar, M., Lukin, M. D., Miller, D. A. B., Monroe, C., Nam, S. W., Narang, P., Orcutt, J. S., Raymer, M. G., Safavi-Naeini, A. H., Spiropulu, M., Srinivasan, K., Sun, S., Vuckovic, J., Waks, E., Walsworth, R., Weiner, A. M., and Zhang, Z. (2019). Development of quantum interconnects for next-generation information technologies.

[Axler 1997] Axler, S. (1997). *Linear Algebra Done Right (2nd ed.).* Springer-Verlag, New York, NY, USA.

[Barnum et al. 2007] Barnum, H., Barrett, J., Leifer, M., and Wilce, A. (2007). Generalized no-broadcasting theorem. *Physical Review Letters*, 99(24).

[Bell and Aspect 2004] Bell, J. S. and Aspect, A. (2004). *Speakable and Unspeakable in Quantum Mechanics: Collected Papers on Quantum Philosophy*. Cambridge University Press, 2 edition.

[Bennett and Brassard 1984] Bennett, C. H. and Brassard, G. (1984). Quantum cryptography: public key distribution and coin tossing. *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, 1(1):175–179.

[Bennett et al. 1993] Bennett, C. H., Brassard, G., Crépeau, C., Jozsa, R., Peres, A., and Wootters, W. K. (1993). Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Physical review letters*, 70(13):1895.

[Bhaskar et al. 2020] Bhaskar, M. K., Riedinger, R., Machielse, B., Levonian, D. S., Nguyen, C. T., Knall, E. N., Park, H., Englund, D., Lončar, M., Sukachev, D. D., and Lukin, M. D. (2020). Experimental demonstration of memory-enhanced quantum communication. *Nature*, 580(7801):60–64.

[Boschi et al. 1998] Boschi, D., Branca, S., De Martini, F., Hardy, L., and Popescu, S. (1998). Experimental realization of teleporting an unknown pure quantum state via dual classical and einstein-podolsky-rosen channels. *Phys. Rev. Lett.*, 80:1121–1125.

[Bourbaki 1989] Bourbaki, N. (1989). *Elements of mathematics, Algebra I*. Springer-Verlag, New York, NY, USA.

[Bouwmeester et al. 1997] Bouwmeester, D., Pan, J.-W., Mattle, K., Eibl, M., Weinfurter, H., and Zeilinger, A. (1997). Experimental quantum teleportation. *Nature*, 390(6660):575–579.

[Bravyi et al. 2006] Bravyi, S., Fattal, D., and Gottesman, D. (2006). Ghz extraction yield for multipartite stabilizer states. *Journal of Mathematical Physics*, 47(6):062106.

[Broadbent and Schaffner 2015] Broadbent, A. and Schaffner, C. (2015). Quantum cryptography beyond quantum key distribution. *Designs, Codes and Cryptography*, 78(1):351–382.

[Cacciapuoti et al. 2020] Cacciapuoti, A. S., Caleffi, M., Tafuri, F., Cataliotti, F. S., Gherardini, S., and Bianchi, G. (2020). Quantum internet: Networking challenges in distributed quantum computing. *IEEE Network*, 34(1):137–143.

[Caleffi and Bianchi 2018] Caleffi, M., C.-A. S. and Bianchi, G. (2018). Quantum internet: From communication to distributed computing! In *Proceedings of the 5th ACM International Conference on Nanoscale Computing and Communication*, NANOCOM'18, New York, NY, USA. Association for Computing Machinery.

[Caleffi 2020] Caleffi, Marcello, C.-A. S. (2020). Quantum switch for the quantum internet: Noiseless communications through noisy channels. *IEEE Journal on Selected Areas in Communications*, pages 1–1.

[Canada 2020] Canada, N. R. C. (2020). Quantum canada survey overview.

[Chakraborty et al. 2019] Chakraborty, K., Rozpedek, F., Dahlberg, A., and Wehner, S. (2019). Distributed routing in a quantum internet.

[Chandra et al. 2018] Chandra, D., Babar, Z., Nguyen, H. V., Alanis, D., Botsinis, P., Ng, S. X., and Hanzo, L. (2018). Quantum topological error correction codes: The classical-to-quantum isomorphism perspective. *IEEE Access*, 6:13729–13757.

[Chen et al. 2018] Chen, X., Cheng, B., Li, Z., Nie, X., Yu, N., Yung, M.-H., and Peng, X. (2018). Experimental cryptographic verification for near-term quantum cloud computing.

[Dahlberg et al. 2019] Dahlberg, A., Skrzypczyk, M., Coopmans, T., Wubben, L., Rozpundefineddek, F., Pompili, M., Stolk, A., Pawełczak, P., Knegjens, R., de Oliveira Filho, J., Hanson, R., and Wehner, S. (2019). A link layer protocol for quantum networks. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM'19, pages 159–173, New York, NY, USA. Association for Computing Machinery.

[Degen et al. 2017] Degen, C., Reinhard, F., and Cappellaro, P. (2017). Quantum sensing. *Reviews of Modern Physics*, 89(3).

[Delteil et al. 2017] Delteil, A., Sun, Z., Falt, S., and Imamoglu, A. (2017). Realization of a cascaded quantum system: Heralded absorption of a single photon qubit by a single-electron charged quantum dot. *Physical Review Letters*, 118(17).

[Deutsch 1996] Deutsch, D. e. a. (1996). Quantum privacy amplification and the security of quantum cryptography over noisy channels. *Phys. Rev. Lett.*, 77:2818–2821.

[Devitt et al. 2013] Devitt, S. J., Munro, W. J., and Nemoto, K. (2013). Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001.

[Di Franco and Ballester 2012] Di Franco, C. and Ballester, D. (2012). Optimal path for a quantum teleportation protocol in entangled networks. *Physical Review A*, 85(1):010303.

[Diamanti et al. 2016] Diamanti, E., Lo, H.-K., Qi, B., and Yuan, Z. (2016). Practical challenges in quantum key distribution. *npj Quantum Information*, 2(1):16025.

[Dodson et al. 2009] Dodson, D., Fujiwara, M., Grangier, P., Hayashi, M., Imafuku, K., ichi Kitayama, K., Kumar, P., Kurtsiefer, C., Lenhart, G., Luetkenhaus, N., Matsumoto, T., Munro, W. J., Nishioka, T., Peev, M., Sasaki, M., Sata, Y., Takada, A., Takeoka, M., Tamaki, K., Tanaka, H., Tokura, Y., Tomita, A., Toyoshima, M., van Meter, R., Yamagishi, A., Yamamoto, Y., and Yamamura, A. (2009). Updating quantum cryptography report ver. 1.

[Dür 2007] Dür, W., B. H. (2007). Entanglement purification and quantum error correction. *Reports on Progress in Physics*, 70:1381–1424.

[Einstein et al. 1935] Einstein, A., Podolsky, B., and Rosen, N. (1935). Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47:777–780.

[Ekert 1991a] Ekert, A. K. (1991a). Quantum cryptography based on bell's theorem. *Phys. Rev. Lett.*, 67:661–663.

[Ekert 1991b] Ekert, A. K. (1991b). Quantum cryptography based on bell's theorem. *Phys. Rev. Lett.*, 67:661–663.

[Elkouss et al. 2020] Elkouss, D., Knegjens, R., and Wehner, S. (2020). The network simulator for quantum information using discrete events.

[European Alliance 2020] European Alliance, Q. I. (2020). Quantum internet alliance.

[Giles 2019] Giles, M. (2019). What is quantum communication? *MIT Technology Review*, 2(1):10.

[Gottesman et al. 2012] Gottesman, D., Jennewein, T., and Croke, S. (2012). Longer-baseline telescopes using quantum repeaters. *Phys. Rev. Lett.*, 109:070503.

[Greenberger et al. 2007] Greenberger, D. M., Horne, M. A., and Zeilinger, A. (2007). Going beyond bell's theorem.

[Guan et al. 2016] Guan, J.-Y., Xu, F., Yin, H.-L., Li, Y., Zhang, W.-J., Chen, S.-J., Yang, X.-Y., Li, L., You, L.-X., Chen, T.-Y., and et al. (2016). Observation of quantum fingerprinting beating the classical limit. *Physical Review Letters*, 116(24).

[Hagouel and Karafyllidis 2012] Hagouel, P. I. and Karafyllidis, I. (2012). Quantum computers: Registers, gates and algorithms. *2012 28th International Conference on Microelectronics - Proceedings, MIEL 2012*.

[Hillery et al. 1999] Hillery, M., Buzek, V., and Berthiaume, A. (1999). Quantum secret sharing. *Physical Review A*, 59(3):1829–1834.

[Holevo 2001] Holevo, A. (2001). *Statistical Structure of Quantum Theory. Lecture Notes in Physics.* Springer, New York, NY, USA.

[Hucul et al. 2014] Hucul, D., Inlek, I. V., Vittorini, G., Crocker, C., Debnath, S., Clark, S. M., and Monroe, C. (2014). Modular entanglement of atomic qubits using photons and phonons. *Nature Physics*, 11(1):37–42.

[Humble et al. 2018] Humble, T. S., Sadlier, R. J., Williams, B. P., and Prout, R. C. (2018). Software-defined quantum network switching. In Blowers, M., Hall, R. D., and Dasari, V. R., editors, *Disruptive Technologies in Information Sciences*, volume 10652, pages 72 – 79. International Society for Optics and Photonics, SPIE.

[IBM 2020] IBM (2020). Ibm quantum experience.

[Inagaki et al. 2013] Inagaki, T., Matsuda, N., Tadanaga, O., Asobe, M., and Takesue, H. (2013). Entanglement distribution over 300 km of fiber. *Optics Express*, 21(20):23241.

[IQT 2020] IQT (2020). Quantum networks: A ten-year forecast and opportunity analysis.

[J. 1998] J., P. (1998). Lectures notes on quantum computation, available at: http://www.theory.caltech.edu/people/preskill/ph229/.

[Jiang et al. 2009] Jiang, L., Taylor, J. M., Nemoto, K., Munro, W. J., Van Meter, R., and Lukin, M. D. (2009). Quantum repeater with encoding. *Physical Review A*, 79(3).

[Jin et al. 2010] Jin, X.-M., Ren, J.-G., Yang, B., Yi, Z.-H., Zhou, F., Xu, X.-F., Wang, S.-K., Yang, D., Hu, Y.-F., Jiang, S., Yang, T., Yin, H., Chen, K., Peng, C.-Z., and Pan, J.-W. (2010). Experimental free-space quantum teleportation. *Nature Photonics*, 4(6):376–381.

[Jozsa 1994] Jozsa, R. (1994). Fidelity for mixed quantum states. *Journal of Modern Optics*, 41(12):2315–2323.

[Kimble 2008] Kimble, H. J. (2008). The quantum internet. *Nature*, 453(7198):1023–1030.

[Kómár et al. 2014] Kómár, P., Kessler, E. M., Bishof, M., Jiang, L., Sørensen, A. S., Ye, J., and Lukin, M. D. (2014). A quantum network of clocks. *Nature Physics*, 10(8):582–587.

[Kumar et al. 2019] Kumar, S., Lauk, N., and Simon, C. (2019). Towards long-distance quantum networks with superconducting processors and optical links. *Quantum Science and Technology*, 4(4):045003.

[Kuzmin et al. 2019] Kuzmin, V. V., Vasilyev, D. V., Sangouard, N., Dür, W., and Muschik, C. A. (2019). Scalable repeater architectures for multi-party states. *npj Quantum Information*, 5(1):115.

[Liang et al. 2019] Liang, Y.-C., Yeh, Y.-H., Mendonça, P. E. M. F., Teh, R. Y., Reid, M. D., and Drummond, P. D. (2019). Quantum fidelity measures for mixed states. *Reports on Progress in Physics*, 82(7):076001.

[Liao et al. 2018] Liao, S.-K., Cai, W.-Q., Handsteiner, J., Liu, B., Yin, J., Zhang, L., Rauch, D., Fink, M., Ren, J.-G., Liu, W.-Y., Li, Y., Shen, Q., Cao, Y., Li, F.-Z., Wang, J.-F., Huang, Y.-M., Deng, L., Xi, T., Ma, L., Hu, T., Li, L., Liu, N.-L., Koidl, F., Wang, P., Chen, Y.-A., Wang, X.-B., Steindorfer, M., Kirchner, G., Lu, C.-Y., Shu, R., Ursin, R., Scheidl, T., Peng, C.-Z., Wang, J.-Y., Zeilinger, A., and Pan, J.-W. (2018). Satellite-relayed intercontinental quantum network. *Phys. Rev. Lett.*, 120:030501.

[Ma et al. 2012] Ma, X.-S., Herbst, T., Scheidl, T., Wang, D., Kropatschek, S., Naylor, W., Wittmann, B., Mech, A., Kofler, J., Anisimova, E., Makarov, V., Jennewein, T., Ursin, R., and Zeilinger, A. (2012). Quantum teleportation over 143 kilometres using active feed-forward. *Nature*, 489(7415):269–273.

[Market 2018] Market, Q. (2018). Quantum computing market & technologies, available at: https://industry40marketresearch.com/reports/quantum-computing-market-technologies.

[McKeown 2009] McKeown, N. (2009). Software-defined networking. *INFOCOM keynote talk*, 17(2):30–32.

[METER et al. 2011] METER, R. V., TOUCH, J., and HORSMAN, C. (2011). Recursive quantum repeater networks. *Progress in Informatics*, 1(8):65–79.

[Mink et al. 2010] Mink, A., Frankel, S., and Perlner, R. (2010). Quantum key distribution (qkd) and commodity security protocols: Introduction and integration.

[Munro et al. 2015] Munro, W. J., Azuma, K., Tamaki, K., and Nemoto, K. (2015). Inside quantum repeaters. *IEEE Journal of Selected Topics in Quantum Electronics*, 21(3):78–90.

[Munro et al. 2013] Munro, W. J., Stephens, A. M., Devitt, S. J., Harrison, K. A., and Nemoto, K. (2013). Quantum communication without the necessity of quantum memories.

[Ndousse-Fetter et al. 2019] Ndousse-Fetter, T., Peters, N., Grice, W., Kumar, P., Chapuran, T., Guha, S., Hamilton, S., Monga, I., Newell, R., Nomerotski, A., Towsley, D., and Yoo, B. (2019). Quantum networks for open science.

[Nielsen 2010] Nielsen, Michael A., C. I. (2010). *Quantum computation and quantum information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA.

[Pant et al. 2019a] Pant, M., Krovi, H., Towsley, D., Tassiulas, L., Jiang, L., Basu, P., Englund, D., and Guha, S. (2019a). Routing entanglement in the quantum internet. *npj Quantum Information*, 5(1):25.

[Pant et al. 2019b] Pant, M., Krovi, H., Towsley, D., Tassiulas, L., Jiang, L., Basu, P., Englund, D., and Guha, S. (2019b). Routing entanglement in the quantum internet. *npj Quantum Information*, 5(1):25.

[Pirandola and Braunstein 2016] Pirandola, S. and Braunstein, S. (2016). Physics: Unite to build a quantum internet. *Nature*, 532:169–171.

[Pirandola et al. 2017a] Pirandola, S., Laurenza, R., Ottaviani, C., and Banchi, L. (2017a). Fundamental limits of repeaterless quantum communications. *Nature Communications*, 8(1):15043.

[Pirandola et al. 2017b] Pirandola, S., Laurenza, R., Ottaviani, C., and Banchi, L. (2017b). Fundamental limits of repeaterless quantum communications. *Nature Communications*, 8(1).

[Pirker and Dur 2019] Pirker, A. and Dur, W. (2019). A quantum network stack and protocols for reliable entanglement-based networks. *New Journal of Physics*, 21(3):033003.

[Proctor et al. 2017] Proctor, T. J., Knott, P. A., and Dunningham, J. A. (2017). Networked quantum sensing.

[Ren et al. 2017] Ren, J.-G., Xu, P., Yong, H.-L., Zhang, L., Liao, S.-K., Yin, J., Liu, W.-Y., Cai, W.-Q., Yang, M., Li, L., Yang, K.-X., Han, X., Yao, Y.-Q., Li, J., Wu, H.-Y., Wan, S., Liu, L., Liu, D.-Q., Kuang, Y.-W., He, Z.-P., Shang, P., Guo, C., Zheng, R.-H., Tian, K., Zhu, Z.-C., Liu, N.-L., Lu, C.-Y., Shu, R., Chen, Y.-A., Peng, C.-Z., Wang, J.-Y., and Pan, J.-W. (2017). Ground-to-satellite quantum teleportation. *Nature*, 549(7670):70–73.

[Rubino et al. 2017] Rubino, G., Rozema, L. A., Massa, F., Araújo, M., Zych, M., ÄŒaslav Brukner, and Walther, P. (2017). Experimental entanglement of temporal orders.

[S. Wehner and Hanson 2018] S. Wehner, D. E. and Hanson, R. (2018). Quantum internet: A vision for the road ahead. *Science*, 362(1):303.

[Sangouard et al. 2009] Sangouard, N., Simon, C., de Riedmatten, H., and Gisin, N. (2009). Quantum repeaters based on atomic ensembles and linear optics.

[Schoute et al. 2016] Schoute, E., Mancinska, L., Islam, T., Kerenidis, I., and Wehner, S. (2016). Shortcuts to quantum network routing.

[Steel 1986] Steel, W. H. (1986). *Interferometry*. Cambridge University Press, Cambridge,UK.

[Takesue et al. 2015] Takesue, H., Dyer, S. D., Stevens, M. J., Verma, V., Mirin, R. P., and Nam, S. W. (2015). Quantum teleportation over 100km of fiber using highly efficient superconducting nanowire single-photon detectors. *Optica*, 2(10):832–835.

[Terhal 2015] Terhal, B. M. (2015). Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2):307–346.

[Turing 1936] Turing, A. (1936). On computable numbers with an application to the entscheidungsproblem. *Proceedings of the London Math Society*, 1(1):230–265.

[U. S. Quantum 2020] U. S. Quantum, National Office, U. S. G. (2020). A strategic vision for america's quantum networks.

[Ursin et al. 2004] Ursin, R., Jennewein, T., Aspelmeyer, M., Kaltenbaek, R., Lindenthal, M., Walther, P., and Zeilinger, A. (2004). Quantum teleportation across the danube. *Nature*, 430(7002):849–849.

[Van Meter 2014] Van Meter, R. (2014). Quantum networking.

[Vardoyan et al. 2019a] Vardoyan, G., Guha, S., Nain, P., and Towsley, D. (2019a). On the capacity region of bipartite and tripartite entanglement switching.

[Vardoyan et al. 2019b] Vardoyan, G., Guha, S., Nain, P., and Towsley, D. (2019b). On the stochastic analysis of a quantum entanglement switch.

[Wang et al. 2014a] Wang, S., Chen, W., Yin, Z.-Q., Li, H.-W., He, D.-Y., Li, Y.-H., Zhou, Z., Song, X.-T., Li, F.-Y., Wang, D., Chen, H., Han, Y.-G., Huang, J.-Z., Guo, J.-F., Hao, P.-L., Li, M., Zhang, C.-M., Liu, D., Liang, W.-Y., Miao, C.-H., Wu, P., Guo, G.-C., and Han, Z.-F. (2014a). Field and long-term demonstration of a wide area quantum key distribution network. *Opt. Express*, 22(18):21739–21756.

[Wang et al. 2014b] Wang, S., Chen, W., Yin, Z.-Q., Li, H.-W., He, D.-Y., Li, Y.-H., Zhou, Z., Song, X.-T., Li, F.-Y., Wang, D., and et al. (2014b). Field and long-term demonstration of a wide area quantum key distribution network. *Optics Express*, 22(18):21739.

[Wiesner 1983] Wiesner, W. (1983). Conjugate coding. *SIGACT News*, 15:77.

[Williams et al. 2019] Williams, B. P., Lukens, J. M., Peters, N. A., Qi, B., and Grice, W. P. (2019). Quantum secret sharing with polarization-entangled photon pairs. *Physical Review A*, 99(6).

[Yin et al. 2017] Yin, J., Cao, Y., Li, Y.-H., Liao, S.-K., Zhang, L., Ren, J.-G., Cai, W.-Q., Liu, W.-Y., Li, B., Dai, H., Li, G.-B., Lu, Q.-M., Gong, Y.-H., Xu, Y., Li, S.-L., Li, F.-Z., Yin, Y.-Y., Jiang, Z.-Q., Li, M., Jia, J.-J., Ren, G., He, D., Zhou, Y.-L., Zhang, X.-X., Wang, N., Chang, X., Zhu, Z.-C., Liu, N.-L., Chen, Y.-A., Lu, C.-Y., Shu, R., Peng, C.-Z., Wang, J.-Y., and Pan, J.-W. (2017). Satellite-based entanglement distribution over 1200 kilometers. *Science*, 356(6343):1140–1144.

[Zaske et al. 2012] Zaske, S., Lenhard, A., KeÃŸler, C. A., Kettler, J., Hepp, C., Arend, C., Albrecht, R., Schulz, W.-M., Jetter, M., Michler, P., and et al. (2012). Visible-to-telecom quantum frequency conversion of light from a single quantum emitter. *Physical Review Letters*, 109(14).

[Zukowski 1993] Zukowski, M., e. a. (1993). Event-ready-detectors bell experiment via entanglement swapping. *Phys. Rev. Lett.*, 71:4287–4290.

# Capítulo 3

# Soft5G+: explorando a softwarização nas redes 5G

Cristiano Bonato Both[1], Kleber Vieira Cardoso[2], Lúcio Rene Prade[1], Victor Hugo L. Lopes[2,3], Ciro J. A. Macedo[2,3]

(1) PPG em Comp. Aplicada – Universidade do Vale do Rio dos Sinos
(2) Instituto de Informática – Universidade Federal de Goiás
(3) Instituto Federal de Educação Ciências e Tecnologia de Goiás

***Resumo***

*O principal objetivo desse minicurso é apresentar o sistema 5G formado por rede de acesso e núcleo, seguindo o conjunto de padrões definidos pelo 3GPP, em especial a partir do 3GPP Release 15, de março de 2018. Nesse contexto, o minicurso terá enfoque prático, mostrando as potencialidades de implementar um sistema 5G a partir da softwarização. O Lançamento (Release) 15 fornecido pela 3GPP introduziu uma nova arquitetura para redes celulares de próxima geração, com base em serviços. Essa Arquitetura Baseada em Serviços (Service-Based Architecture – SBA) separa estruturalmente os planos de dados e controle, com a inserção de virtualização e de fatiamento de recursos para um alto nível de desacoplamento entre as principais funções da rede de acesso e do núcleo. Assim, essa inovadora e disruptiva arquitetura oferece grande flexibilidade para o desenvolvimento e implantação de redes de próximas gerações, principalmente considerando à migração de sofisticados recursos de hardware para componentes de software executados em sistemas de nuvem. O minicurso visa apresentar projetos e implementações de componentes de software que são utilizados na rede de acesso e no núcleo. Por exemplo, os autores demonstrarão projetos de código aberto que permitem a implementação de uma rede de acesso atual, i.e., LTE/4G (projeto srsLTE), assim como sua integração com um núcleo 5G (projeto free5GC). Além disso, os autores apresentarão a utilização desses componentes de software em contêineres para explorar os serviços nativos de sistemas de nuvem no contexto 5G. Por fim, o minicurso mostrará as perspectivas e potencialidades dessa nova geração de redes que tem como principais características a flexibilidade e a programabilidade dos seus componentes. Para ilustrar essas características, os autores apresentarão abordagens para integração de redes sem fio não-3GPP para dispositivos IoT (e.g., LoRa) com uma rede 3GPP.*

***Abstract***

*The main goal of this short course is to present the 5G system composed of the access network and the core, following the set of standards defined by the 3GPP, especially from the 3GPP Release 15, of March 2018. In this context, the short course will have a practical focus, showing the potential of implementing a 5G system from software. The Release 15 provided by 3GPP introduced a new architecture for next generation cellular networks, based on services. This Service-Based Architecture (SBA) separates control and data plans, and adds virtualization and resource slicing in order to achieve a high level of decoupling between the main functions from the access network and the core. Thus, this innovative and disruptive architecture offers great flexibility for the development and deployment of next-generation networks, especially considering the migration from sophisticated hardware resources to software components running on cloud systems. The short course aims to present projects and implementations of software components that are used in the access network and in the core. For example, the authors will present open source projects that allow the implementation of a current access network, that is, LTE/4G (srsLTE project), as well as its integration with a 5G core (free5GC project). In addition, the authors present the use of these software components in containers to exploit the native services of cloud systems in the 5G context. Finally, the short course aims to present the perspectives and potentials of this new generation of networks that have as main characteristics the flexibility and programmability of their components. To illustrate these characteristics, the authors present approaches for the integration of non-3GPP wireless networks for IoT devices (e.g., LoRa) with a 3GPP network.*

## 3.1. Introdução

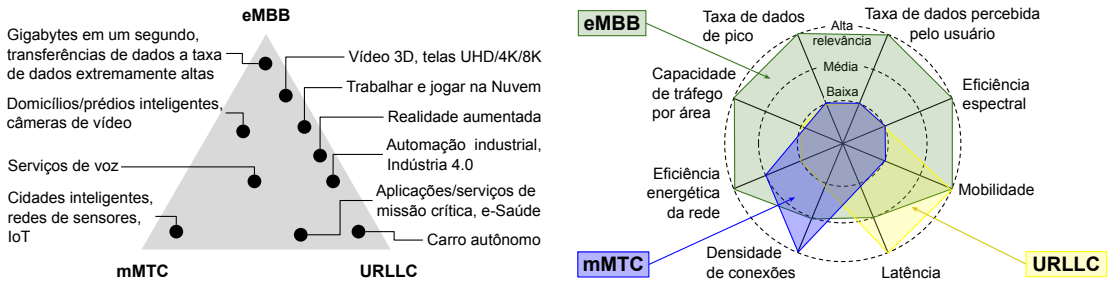
Antes da quinta geração, as redes móveis sem fio se focaram predominantemente no aumento da taxa de transmissão de dados e na oferta de acesso para os consumidores finais. Entretanto, mais do que apenas melhorar a largura de banda e reduzir a latência, redes 5G [1] permitem que soluções verdadeiramente disruptivas surjam em todos os tipos de indústrias [2]. Embora muitos países, como o Brasil, ainda estejam discutindo e se preparando para a implantação de redes 5G, desde 2019 vários países (*e.g.*, China, Coreia do Sul e Estados Unidos) já estão implantando, testando e/ou efetivamente utilizando a mais recente geração de redes móveis sem fio. A expectativa é que mais de 2,4 bilhões de dispositivos utilizem redes 5G até 2025 [3]. Por enquanto, taxas próximas (e até superiores) a 1 Gbps ainda são a principal novidade, mas vários serviços, sobretudo baseados em IoT, começam a ser planejados e implantados. Carros autônomos, Indústria 4.0 e realidade virtual/aumentada/mista são algumas das apostas e costumam servir como casos de uso interessantes para as novas funcionalidades da rede. No entanto, como nas gerações anteriores, é difícil antever com acurácia quais aplicações ou serviços serão efetivamente adotados. Por enquanto, o único consenso é que redes 5G, quando estiverem implantadas em uma escala significativa, representarão um novo passo para as comunicações sem fio e terão um impacto na sociedade tão (ou mais) profundo quanto as gerações anteriores [4].

De fato, a quinta geração representa um salto tecnológico notável em relação à quarta geração, introduzindo inovações de hardware e, sobretudo, de software significativos. É importante esclarecer que esse salto na verdade esconde os vários passos inter-

mediários que o compõe. Na indústria de telecomunicações, inclusive por questões de publicidade, se acumula uma quantidade representativa de evoluções e novidades tecnológicas antes de definir uma geração, o que não foi diferente com redes 5G. Para ilustrar, vamos tomar como o exemplo os Lançamentos (*Releases*) da 3GPP (*3rd Generation Partnership Project*) que é uma das principais organizações de padronização de redes móveis sem fio da atualidade. Redes 4G/LTE (*Long-Term Evolution*) foram introduzidas em 2008 no Lançamento 8 [5] e, desde então, passaram a receber várias atualizações (representadas por novos Lançamentos) até que redes 5G fossem introduzidas oficialmente no Lançamento 15 [6] em 2018. Por outro lado, há fatores técnicos que também apoiam a definição de cada geração. Em redes 5G, isso aparece no uso do espectro que, além de ampliar a utilização das bandas tradicionalmente alocadas na ordem de centenas de MHz até algumas unidades de GHz, introduz ainda a utilização de bandas de dezenas de GHz. Além disso, redes 5G consolidam um intenso processo de *softwarização* que se destaca pela adoção de sistemas de nuvem e tecnologias como virtualização [7], redes definidas por software [8], fatiamento de rede (*Network Slicing*) [9] e arquitetura de software baseada em serviços [10] (e microsserviços).

Além dos aspectos tecnológicos, redes 5G também introduzem mudanças no modelo de negócios das empresas. Tradicionalmente, operadores de sistemas de comunicação móvel sem fio se focavam em usuários finais como principais fontes de receita. Em redes 5G, a intenção é ampliar (ou até migrar) o foco para ter indústrias como principais clientes [11, 12]. Para dar suporte a essa ampliação do modelo de negócio das empresas de telecomunicações, três requisitos (ou cenários) principais foram definidos: Banda larga móvel aprimorada (eMBB – *Enhanced Mobile Broadband*), Comunicações de baixa latência ultra confiáveis (URLLC – *Ultra-Reliable Low-Latency Communication*) e Comunicações massivas de tipo de máquina (mMTC – *Massive Machine Type Communications*). Esses cenários podem ser ilustrados pela Figura 3.1(a) que foi introduzida pela ITU (*International Telecommunication Union*) [13], em 2015, como parte da visão do que seriam redes 5G. Essa ilustração, assim como a Figura 3.1(b), que apresenta as capacidades principais de cada cenário, são amplamente utilizadas para tentar resumir algumas propriedades relevantes de redes 5G.




(a) Cenários 5G (b) Capacidades dos cenários

**Figura 3.1. Cenários (ou requisitos) definidos para as redes 5G (à esquerda), assim como as capacidades a serem ofertadas nesses cenários (à direita).**

Para atender os requisitos dos diferentes cenários, os conceitos de softwarização de rede passaram a ser adotados, tanto pela academia quanto pela indústria, para tornar

viável o desenvolvimento de redes (ou sistemas) 5G nos últimos anos e também sua evolução [9]. O principal motivo para a introdução intensiva de software em sistemas 5G é ampliar a flexibilidade à arquitetura de rede móvel, suportando diferentes requisitos em desafiadores cenários que estão surgindo na sociedade digitalizada. Além disso, a softwarização de rede permite usufruir da computação em nuvem tradicional, assim como de suas variantes de menor escala, *i.e.*, *fog* e *edge*. Nesse contexto, as funções de rede de acesso por rádio e núcleo passam a ser implantadas utilizando as vantagens da computação em nuvem, como armazenamento farto, processamento em larga escala, elasticidade, etc. Além de habilitar novos serviços, no longo prazo, a softwarização reduz os custos de implantação e de operação de sistemas 5G.

Dado o enfoque do minicurso na quinta geração de redes móveis celulares e sua evolução, definimos uma abordagem para descrever as informações e estruturar o texto conforme descrito a seguir. Inicialmente, o interesse está nas redes móveis para transporte de dados, embora a primeira e segunda geração dessas redes tivessem como fundamento a transmissão de voz. Adicionalmente, assumimos que uma rede móvel celular está organizada em rede de acesso por rádio e núcleo, embora essa divisão não seja tão bem definida até a terceira geração. Por fim, abordamos as redes móveis celulares sob a perspectiva de dispositivos de *hardware* padronizados que são controlados por componentes de software. Em geral, tanto o *hardware* quanto o *software* seguem especificações abertas que garantem sua interoperabilidade. Vale destacar que esse não é o cenário observado em redes anteriores à quarta geração, mas tende a ser a abordagem a ser seguida, a partir da quinta geração. Além disso, a quinta geração inaugura a unificação de padrões, a qual teve início com a consolidação da LTE, mas não era uma realidade no início da quarta geração, disputada também pela tecnologia WiMAX (*Worldwide Interoperability for Microwave Access*). Baseado na evolução das redes móveis, o minicurso também tem como objetivo apresentar as potencialidades de implementar um sistema 5G a partir da softwarização, seguindo os últimos Lançamentos da 3GPP. Adicionalmente, o minicurso introduz os principais projetos e implementações abertas de componentes de software que estão sendo utilizados em rede de acesso e núcleo, explorando os serviços nativos em nuvem para o contexto 5G. Por fim, discutimos as perspectivas dessa nova geração de redes que tem como principais características a flexibilidade e programabilidade dos seus componentes.

## 3.2. Redes móveis celulares até a quarta geração

Os primeiros sistemas de telefonia móvel empregavam um transceptor de alta potência, evoluindo para um esquema de organização com múltiplos transceptores (de menor potência) que permitiu aumentar a capacidade em termos de cobertura e reaproveitamento de recursos. Basicamente, essa é a essência das redes celulares, isto é, a sua estruturação em um conjunto de células, cada uma atendendo a uma pequena área geográfica com o uso de uma torre própria composta de antenas com transmissores de baixa potência. Para cada célula é alocada uma banda de frequências e a torre funciona como estação rádio base (transmissor, receptor e uma unidade de controle). Além de favorecer a forma de operação do sistema como um todo, auxiliando nos problemas relacionados aos efeitos de propagação dos sinais, outro benefício dessa forma de organização celular é o reuso de frequências. A coordenação entre o conjunto de células permite o reuso de uma

mesma frequência em células próximas não adjacentes, realizando o controle da potência de transmissão e a alocação de canais [14].

### 3.2.1. Operação dos sistemas celulares

Mesmo observando que os sistemas celulares evoluíram consideravelmente desde a sua proposição, a essência do seu modo de operação não sofreu grandes modificações. A estação rádio base transceptora (BTS - *Base Transceiver Station*, ou simplesmente BS) contida em cada célula é composta de uma ou mais antenas, um controlador e um conjunto de transceptores para a comunicação através dos canais atribuídos à célula. Nesse contexto, o controlador é responsável por todo o processo de comunicação entre os diversos dispositivos de usuário (UE – *User Equipment*) ativos dentro da célula e o restante da rede. A BS opera de forma automatizada, sem necessidade de ações por parte do usuário e disponibiliza dois tipos de canais, *i.e.*, controle e tráfego. Os canais de controle sendo utilizados para trocar informações de configuração, tais como aquelas necessárias à inicialização e manutenção de chamadas para a associação dos UEs às BSs próximas. Os canais de tráfego são utilizados para transmitir efetivamente os dados e voz dos usuários.

Originalmente, cada BS possuía conexão com o MTSO (*Mobile Telecommunications Switch Office*), que por sua vez, conectava-se à rede telefônica pública comutada (PSTN - *Public Switched Telephone Network*). Um único MTSO servia a várias BSs realizando tarefas de encaminhamento de chamadas à usuários fixos e móveis, monitoramento de controle e tarifação, manutenção de chamadas em andamento, além de *paging* (o procedimento utilizado para a localização da célula em que um assinante se encontra, antes de uma solicitação de início de chamada ser encaminhada) e *handoff* (o processo de transferência de uma chamada em andamento, ou uma sessão de dados, de uma célula à outra). Cabia às BSs o envio periódico de sinais de sincronização, permitindo aos UEs escanearem, selecionarem e monitorarem os sinais dos canais de controle disponíveis. Esses sinais periódicos auxiliavam no processo de associação dos UEs à BS mais adequada. Sempre que uma associação ocorria, um registro era realizado no MTSO, o qual controlava a célula e permitia identificar a localização de cada UE.

Na subseção a seguir, mostramos como as redes celulares evoluíram desde a primeira geração. Naturalmente, houve várias alterações e a introdução de diversas tecnologias. No entanto, a maior parte dos conceitos introduzidos no início das redes celulares ainda persiste, tais como célula, reuso de frequências, *paging*, *handoff*, dentre outros.

### 3.2.2. A evolução das redes móveis

Os sistemas de redes móveis têm evoluído de forma significativa desde que foram introduzidos, em meados dos anos 1980. Conforme descrito anteriormente, as evoluções são agrupadas sob o conceito de *gerações*, marcadas por um conjunto relevante novas tecnologias que alteram a oferta de serviços pela rede móvel. Apesar da quinta geração já estar entrando em operação comercial, vamos apresentar brevemente algumas características que marcaram cada uma das gerações anteriores, conforme sumarizado na Tabela 3.1.

E interessante observar que, ainda hoje, apenas a primeira geração das redes móveis não tem nenhuma operação comercial. Ou seja, todas as demais gerações ainda possuem equipamentos em operação, pelo menos como parte de redes mais modernas.

**Tabela 3.1. Evolução das gerações: algumas características importantes [14, 15, 16].**

| | 1G | 2G | 3G | 4G |
|---|---|---|---|---|
| Implementação | ≈ 1984 | ≈ 1991 | ≈ 1999 | ≈ 2009 |
| Serviços | Voz analógica | Voz digital<br>SMS | Pacotes de dados<br>de alta capacidade | Comunicação IP |
| Taxa de dados | 1.9 kbps | 14.4 kbps | 384 kbps | 200 Mbps |
| Multiplexação | FDMA | TDMA, CDMA | TDMA, CDMA | OFDMA, SC-FDMA |
| Rede de acesso | AMPS | GSM, PDC, IS-95,<br>IS-136, EDGE, GPRS | CDMA 2000, UMTS,<br>TD-SCDMA,WCDMA | LTE, WiMAX |
| Núcleo da rede | PSTN | PSTN - SS | PSTN - SS, rede de pacotes | EPC |
| 3GPP | - | - | Lançamento 1999 | Lançamento 8 |

Analisando as evoluções das gerações, podemos compreender o aprimoramento das características arquiteturais (rede de acesso e núcleo) e o crescente emprego de soluções de *software* nessas novas gerações. Além disso, é importante contextualizar a origem do sistema analógico baseado em circuitos (1G), fortemente acoplado ao sistema telefônico cabeado, tornando-se parcialmente orientado a pacotes (2G e 3G) e, finalmente, oferecendo um suporte completo à comunicação de pacotes IP (4G) [16]. Cada uma dessas gerações é descrita brevemente a seguir com o objetivo de auxiliar na compreensão do atual estágio que estamos vivendo, em especial, as novidades introduzidas efetivamente na quinta geração.

### Primeira Geração - 1G

A primeira geração (1G) dos sistemas de redes móveis sem fio foi criada para oferecer serviço de comunicação de voz analógica de forma a estender a rede telefônica comutada (PSTN). Nesse contexto, o AMPS (*Advanced Mobile Phone Service*) foi o padrão de rede de acesso 1G mais amplamente implementado. Apesar de não haver consenso sobre uma clara distinção entre rede de acesso e núcleo nesta arquitetura, observa-se que o MTSO pode ser considerado como núcleo, pois realiza tarefas de controle de *handover*, registro e autenticação de UEs, roteamento e gestão de ligações telefônicas (*paging*, *handoff*, etc.). Além disso, o MTSO interliga a PSTN às múltiplas BTSs [17], que por sua vez realizam as conexões com UEs, através de canais analógicos em modulação por frequência (FM - *Frequency Modulation*) e acesso múltiplo aos poucos canais disponíveis via FDMA (*Frequency Division Multiple Access*), em um espectro licenciado. Apesar de apresentar várias limitações (como canais de baixa capacidade e taxa de dados, limitação do uso da banda pela adoção de modulação única para voz e dados de controle, além de falta de criptografia), um importante caminho foi aberto levando em direção às próximas gerações.

### Segunda Geração - 2G

O sistema 2G foi introduzido no início dos anos 90 e apresentou suporte a modulações digitais, apesar de ainda continuar baseado na transmissão de dados com comutação por circuitos, em sua primeira versão. Essa técnica de modulação proporcionou a oferta de serviços de voz digital e pacotes de dados básicos, com serviço de mensagens curtas

de texto (SMS - *Short Message Service*). O emprego de novas modulações, com canais ofertando uma maior capacidade, bem como a inclusão de novas técnicas de detecção e correção de erros e algoritmo de compressão e descompressão [18], melhorou a qualidade de serviço (QoS - *Quality of Service*) para os usuários finais. Além disso, maiores taxas de transmissão de dados (Tabela 3.1) e eficiência espectral foram alcançadas com a introdução de métodos de acesso dinâmico, como o acesso múltiplo por divisão de tempo (TDMA - *Time Division Multiple Access*) e por divisão de código (CDMA - *Code Division Multiple Access*) [14].

Toda a família do sistema 2G ainda está operacional hoje em dia, sendo que importantes evoluções foram implementadas para dar suporte ao incremento da demanda por serviços de dados. O Sistema Global de Comunicações Móveis (GSM - *Global System for Mobile Communications*) é a primeira versão implementada, seguido pela 2,5G, o qual utiliza o padrão GPRS (*General Packet Radio Service*), introduzindo o suporte à comutação por pacotes. Essa evolução foi finalizada com a chamada geração 2,75G, através da tecnologia EDGE (*Enhanced Date Rates For GSM Evolution*), que apresentou um padrão de melhoria nas taxas de dados para o GSM [19, 18]. Já na segunda geração, podemos observar a dificuldade em definir as fronteiras e a qual geração atribuir cada inovação.

Sob o ponto de vista arquitetural, o sistema 2G, ilustrado na Figura 3.2, é composto pelo sistema de estações rádio base (BSS -*Base Station System*), com o papel de RAN (*Radio Access Network*) para os usuários móveis GSM (MS - *Mobile Station*) e o sistema de comutação (SS - *Switching System*) como o núcleo da rede [20, 21]. O BSS é composto de diversas BTSs interligadas e coordenadas por controladores de estações rádio base (BSC - *Base Station Controller*). O SS encontra-se na camada da rede, com a responsabilidade de gerenciar o BSS, provendo a comutação de chamadas (PSTN) através do centro de comutação móvel (MSC - *Mobile Switching Center*), e a comutação dos pacotes para a rede de dados por intermédio do nó de suporte ao tráfego de dados GPRS (SGSN - *Serving GPRS Support Node*). Além disso, o SS realiza serviços de registro e autenticação de usuários locais e visitantes (HLR *Home Location Register*, VLR - *Visitor Location Register* e AUC - *Authentication Center*) e o registro de identificação de equipamentos (EIR - *Equipment Identity Register*). Toda a comunicação externa (PSTN e ISDN - Integrated Services Digital Network) é realizada por intermédio do *gateway* do centro de comutação móvel (GMSC - *Gateway Mobile Switching Center*) [22].

### Terceira Geração - 3G

Os benefícios introduzidos pela segunda geração ampliaram a implantação global e o interesse do público geral em redes móveis. Desta forma, novas demandas foram geradas, tanto para a academia, quanto para a indústria, e tais esforços culminaram na terceira geração (3G). Essa geração foi introduzida no início do novo milênio para ofertar taxas de dados ainda maiores para a época, com melhor QoS e cobertura do sinal. Além disso, 3G consolidou o suporte à comunicação sem fio com altas taxas de transmissão para serviços multimídia, *i.e.*, dados, voz e vídeo. Entre muitos padrões 3G, o CDMA de banda larga (WCDMA - *Wideband* CDMA) foi amplamente adotado [23]. Os requisitos definidos para essa geração foram descritos em uma iniciativa da ITU no ano 2000 (IMT-

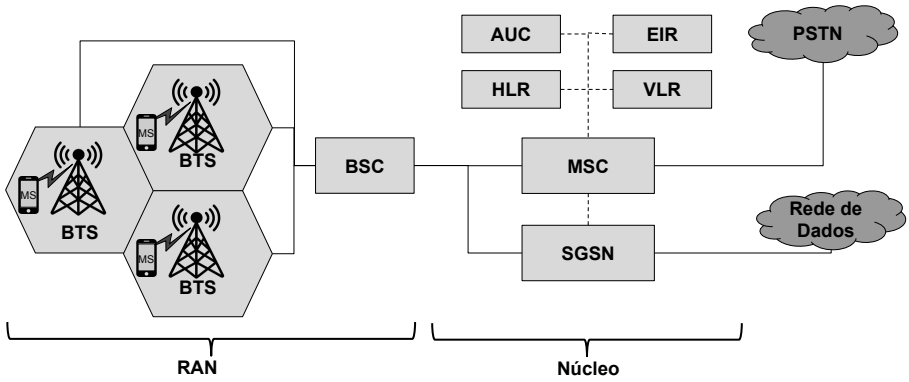



**Figura 3.2. Visão geral da arquitetura GSM/GPRS (adaptado de [20]).**

2000 - *International Mobile Telecommunications*), na qual se destacam:

- Qualidade de voz comparável à rede telefônica pública comutada;
- Taxa de transmissão de dados de 144 kbps para usuários em altas velocidades em áreas amplas (veículos automotores);
- 384 kbps de taxa de transmissão de dados para pedestres com baixa mobilidade em áreas pequenas;
- Suporte para serviços comutados por pacotes e por circuitos;
- Interface adaptativa que permite assimetria entre tráfego enviado e recebido para a rede de dados (Internet);
- Melhorias na eficiência espectral;
- Suporte para uma ampla variedade de dispositivos móveis;
- Flexibilidade para a inclusão de novos serviços e tecnologias.

Surgida com o propósito de especificar completamente esta geração, a 3GPP consolida-se como importante organização de padronização, em que as especificações são organizadas em documentos chamados lançamentos (*releases*), frutos de esforços dos seus 3 diferentes grupos de especificações técnicas: 1) RAN, 2) Serviços e Aspectos dos Sistemas e o Núcleo da Rede e 3) Terminais de Usuários. A terceira geração foi inicialmente especificada no lançamento 1.999, no qual as alterações arquiteturais necessárias ao atendimento dos requisitos foram apresentadas [24]. Posteriormente, outras importantes alterações foram apresentadas nos Lançamentos 5 e 6 (3,5G). O subsistema da rede de rádio (RNS - *Radio Network Subsystem*) que forma a RAN terrestre UMTS (ULTRAN) é composta de estações rádio base do tipo Nó B (NodeB), as quais substituem as BTSs da geração anterior, controladas por RNCs (*Radio Network Controller*). Mesmo não havendo uma alteração arquitetural tão significativa em relação à 2G, um importante requisito básico do Lançamento 99 é o fornecimento de acesso por parte da ULTRAN à infraestrutura

GSM/GPRS existente sem grandes impactos ao núcleo da rede [25], mantendo a coexistência dos comutadores por circuitos e por pacotes.

### Quarta Geração - 4G

A melhoria dos sistemas de comunicação sem fio em conjunto com uma significativa evolução e popularização dos dispositivos móveis de usuários, especialmente *smartphones* e *tablets*, impulsionaram a indústria de telecomunicações no final da primeira década deste século. Esses dispositivos passaram a contar com bons recursos computacionais e contribuíram para um ininterrupto crescimento da demanda por serviços de banda larga móvel, com altas taxas de dados e QoS. Essa grande demanda abriu o caminho para uma nova geração de padrões e sistemas. A demanda fez com que a 3GPP iniciasse dois projetos em paralelos, (*i*) a Evolução de Longo Prazo (LTE), ou E-UTRAN (*Evolved Universal Terrestrial Access Network*) [26], com foco na definição de uma nova RAN e (*ii*) a Evolução da Arquitetura do Sistema (SAE - *System Architecture Evolution*), para a proposição de um novo núcleo da rede. Baseado nesses projetos, surgiu a arquitetura EPS (*Evolved Packet System*), com suas especificações iniciais publicadas pela 3GPP no Lançamento 8 em 2009 [5]. Esse lançamento teve como objetivo atender às diretivas apontadas pela ITU [27], tais como:

- Ser integralmente baseado em comutação por pacotes IP;
- Suportar taxas de dados de pico de até aproximadamente 100 Mbps, para acesso móvel de alta mobilidade, e 1 Gbps, para acesso móvel de baixa mobilidade;
- Incrementar a capacidade de atendimento de usuários por célula com uso de recursos de rede compartilhados;
- Suportar a conexão simultânea a duas ou mais células durante uma conexão para oferecer transição suave (*smooth handover*) em redes heterogêneas;
- Suportar aplicações multimídia de próxima geração de alta qualidade.

Dois padrões emergiram dos esforços globais para padronização da 4G, sendo o LTE desenvolvido pela 3GPP e o IEEE 802.16 desenvolvido por um grupo de trabalho do IEEE (*Institute of Electrical and Electronics Engineers*), também chamado de WiMAX, conforme sugerido pelo consórcio de seus fabricantes. O grupo de trabalho do IEEE 802.16 foi motivado pelo enorme sucesso do IEEE 802.11 (ou WiFi), para a comunicação sem fio em redes locais. Entretanto, WiMAX teve como objetivo a operação em comunicações sem fio fixas de alta velocidade (de dados), de forma a satisfazer as necessidades da 4G. Os dois padrões se mostraram equivalentes em termos de desempenho e tecnologia [14], baseados em acesso múltiplo via multiplexação ortogonal por divisão de frequência (OFDMA - *Orthogonal Frequency Division Multiplexing / Multiple Access*). Desta forma, o WiMAX assumiu um importante papel como tecnologia de acesso sem fio de banda larga fixa, enquanto o LTE se tornou o padrão universal para a interface aérea da 4G. A escala alcançada pelo LTE e a sua capacidade de oferecer um serviço semelhante

ao WiMAX em redes sem fio de banda larga fixa são alguns fatores que tornaram o padrão da 3GPP dominante na 4G.

O padrão LTE define uma rede de acesso composta por estações rádio base do tipo NodeB evoluída (eNB - *evolved NodeB*), as quais são interligadas entre si e com o núcleo EPC. Uma rede LTE implementa modulações de alta ordem (64 QAM ou superior) com alta largura de banda (superior a 10 MHz), com multiplexação espacial em *downlink* (4x4 ou superior), oferecendo uma grande melhoria na taxa de transmissão de dados e, consequentemente, no desempenho da recepção (conforme resumido na Tabela 3.1). Além das grandes mudanças na interface aérea, outro aspecto disruptivo introduzido no sistema 4G é a arquitetura de rede plana de rádio e núcleo. Essa arquitetura permite inserir menos dispositivos intermediários e uma estrutura menos hierárquica para a rede, sem a existência de controladores centralizados. Desta forma, reduz a latência nos processos de conexões dos UEs e no tempo necessário para o *handover*, além de gerar outras facilidades, como simplificação da camada de enlace. Essa característica de uma arquitetura plana permitiu a expansão de serviços de telefonia móvel para as redes de pacotes IP [28], também abrindo caminho para a introdução da softwarização nesses sistemas. A Figura 3.3 ilustra uma visão geral da evolução arquitetural desde a 2G, na qual podemos observar a evolução na rede de acesso, o método de acesso dinâmico empregado, bem como a dependência do controlador centralizado e o tipo de comutação no núcleo da rede.

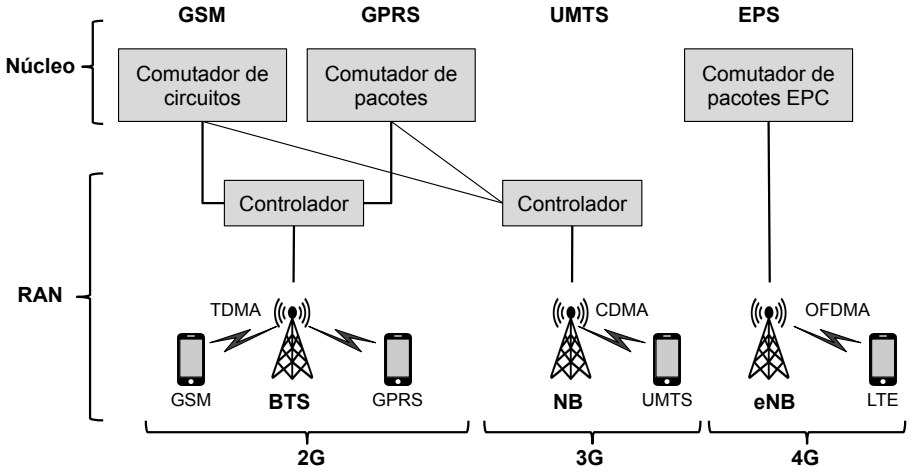



**Figura 3.3. Visão geral da evolução arquitetural das redes móveis (adaptado de [26]).**

### LTE-*Advanced*

Devido à continua demanda por melhorias nas redes móveis, a 3GPP apresentou no Lançamento 10 avanços na interface aérea e na arquitetura de núcleo, dando origem ao LTE-*Advanced* (ou simplesmente LTE-A) [29]. A Figura 3.4 apresenta os elementos da configuração do LTE-A e os planos de dados e controle. Além disso, nessa arquitetura, cada *eNodeB* implementa um controlador próprio (BBU - *Baseband Unit*), mantendo o EPC com a responsabilidade de interconectar todos os elementos arquiteturais. A funcionalidade dos componentes principais do EPC são descritas a seguir.

- HSS - *Home Subscriber Server*: mantém um banco de dados que contém informa-

ções de usuários e UEs, dando suporte às funções de gerenciamento de mobilidade, configuração de sessões e chamadas, autenticação e autorização de acesso.

- PCRF - *Policy and Charging Rules Function*: provê o suporte para a detecção de fluxo de dados para aplicação de políticas e cobranças de serviços de dados.
- MME - *Mobility Management Entity*: entidade de gerenciamento de mobilidade que lida com o controle de sinalização para rastreamento, controle de segurança e *paging* de UEs em modo ativo.
- S-GW - *Serving Gateway*: lida com os pacotes de dados IP enviados e recebidos pelos UEs, sendo o ponto de interconexão entre RAN e EPC, para o fluxo de dados entre as *eNodeBs*.
- P-GW - *Packet Data Network (PDN) Gateway*: é o ponto de interconexão entre EPC e as redes IP externas, roteando os pacotes, além de lidar com a alocação de endereço IP e controle de políticas de tarifação.

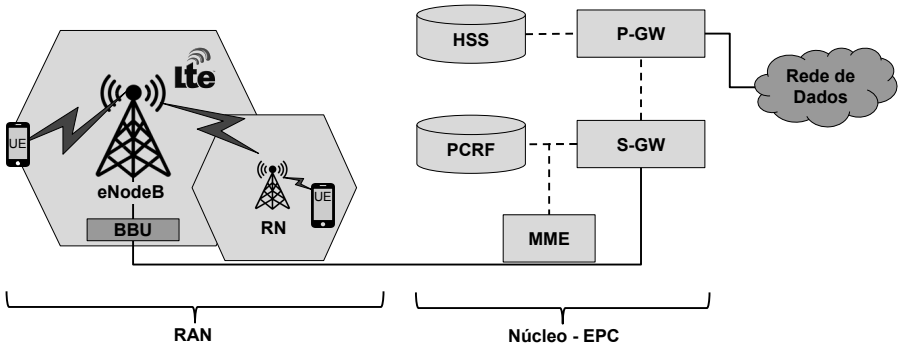



**Figura 3.4. Elementos da arquitetura LTE-*Advanced* (adaptado de [14]).**

O LTE-A continuou sendo aprimorado em outros lançamentos da 3GPP, cujo objetivo foi fornecer taxas de *bits* mais altas de maneira econômica e, ao mesmo tempo, cumprir completamente os requisitos estabelecidos pela ITU para a 4G. Entre esses requisitos está uma maior taxa de dados de pico (tanto para *uplink* quanto para *downlink*), maior eficiência espectral, maior número de UEs ativos simultaneamente e melhor desempenho nas bordas da célula. Outros avanços do LTE-A incluem o emprego de agregação de portadoras e melhorias nas técnicas de múltiplas antenas (MIMO - *Multiple-Input and Multiple-Output*) para suportar nós secundários retransmissores (RN - *Relay Nodes*).

A inclusão dos RNs visou resolver a queda da qualidade do sinal e taxas de transmissão nas bordas das células, causadas pela redução do nível do sinal e incremento de interferências. De uma forma simplificada, um RN é uma célula de menor raio de operação, distribuída nas regiões de fronteira das células principais. É importante destacar que não se trata de um simples repetidor de sinal, pois tem a capacidade de demodular e decodificar os sinais da *eNodeB* (neste contexto, chamada de *donor eNodeB*), aplicando correções de erros, quando necessário, operando tanto em *donwlink* quanto em *uplink*.

Outras importantes evoluções foram incluídas ao LTE-*Advanced*, onde destacamos as alterações arquiteturais promovidas pelo Lançamento 14 da 3GPP [30]. Nessa atualização, foi introduzida a EPC-CUPS (*Control and User Plane Separation*), ilustrada na Figura 3.5 (parte direita da imagem) em comparação com a arquitetura EPC tradicional (parte esquerda da imagem). No entanto, CUPS também pode ser considerada uma funcionalidade da arquitetura 5G Não-Autônoma (NSA – *Non-Stand Alone*), a qual será apresentada na próxima seção. Portanto, descreveremos CUPS e outras evoluções posteriormente, no contexto da arquitetura 5G NSA. Vale relembrar que as gerações das redes móveis celulares possuem diferenças claras, mas suas fronteiras não são bem definidas quando observamos as evoluções introduzidas em cada lançamento 3GPP, especialmente nos que marcam uma transição (como o Lançamento 14).

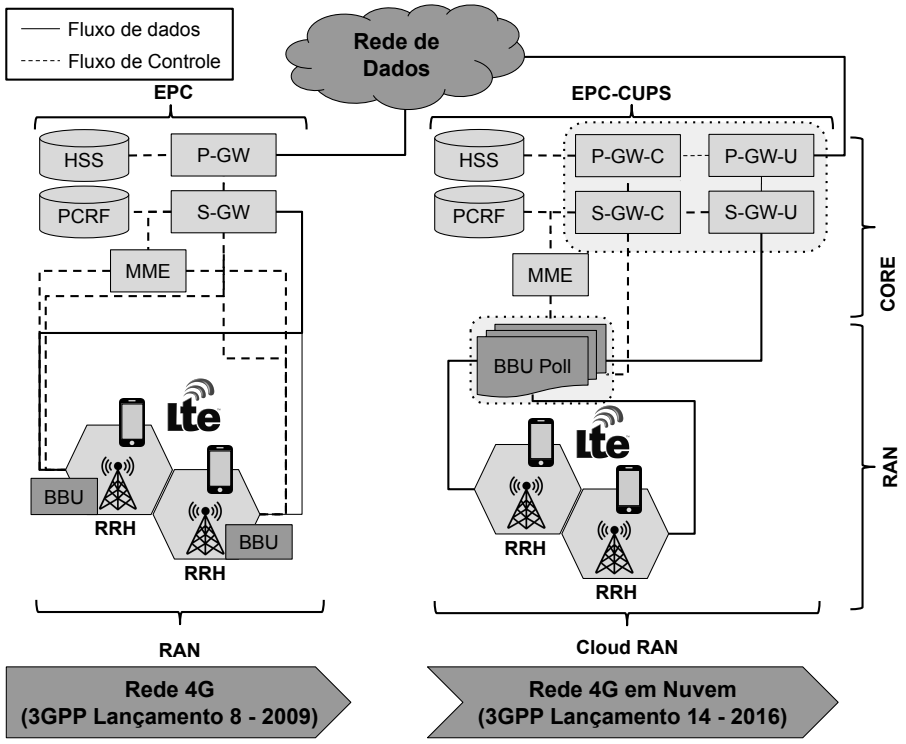



**Figura 3.5. Evolução arquitetural do 4G.**

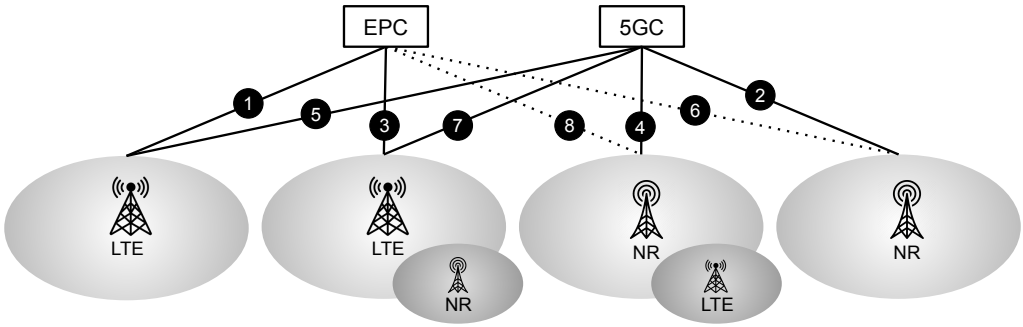
## 3.3. Redes 5G e posteriores

A evolução das redes móveis celulares sempre buscou atingir um rádio onipresente de alta capacidade, *i.e.*, com níveis cada vez mais baixos de latência e maior capacidade de transmissão. Em um primeiro momento, a chegada da 5G adiciona a essa evolução a abertura de um caminho para tecnologias de IoT e comunicação inteligentes, explorando novas formas de conectividade entre veículos e infraestrutura (V2I - *Vehicle-to-Infrastructure*), veículo para veículo (V2V - *Vehicle-to-Vehicle*), conexão pessoa-a-pessoa (D2D - *Device-to-Device*). Entretanto, essa evolução significativa e amplamente publicizada esconde a verdadeira revolução de 5G, a qual descrevemos a seguir.

5G está sendo desenvolvida para inovação e evolução dos sistemas celulares. Essa

geração é a mais dinâmica e flexível das redes celulares já projetada, fazendo uso extensivo de aplicativos e núcleo nativos em nuvem. Além disso, 5G tem potencial para ser moldada durante seu desenvolvimento, a fim de absorver novos saltos evolutivos dentro do mesmo padrão. Essa característica visa acompanhar a evolução exponencial das principais tecnologias, como inteligência artificial e automação, por exemplo. Para atingir essa maturidade, 5G foi projetada para fazer amplo uso da virtualização de serviços e microsserviços. Em um primeiro momento, 5G será operada por uma rede híbrida, na qual a convergência, a integração e a coexistência com sistemas legados é inevitável. Entretanto, no futuro, o sistema 5G será predominantemente composto de funções de redes virtuais.

A 3GPP projetou a rede de acesso por rádio 5G (*NG-RAN – Next Generation - Radio Access Network*) e o núcleo 5G baseado em serviços (SBA - *Service-Based Architecture*) de forma independente e interoperável. Baseado nesse projeto, trata-se de uma abordagem diferente das gerações anteriores de redes móveis, nas quais RAN e núcleo foram projetados de maneira acoplada. Assim, a partir da 5G, o padrão prevê a integração de elementos de diferentes gerações em diferentes configurações. Considerando especificamente a transição entre 4G e 5G, durante a fase inicial da definição da arquitetura 5G, foi definida uma terminologia para identificar as diferentes variantes que surgiram da integração entre rede de acesso e núcleo. As opções definidas estão ilustradas na Figura 3.6.




| Rede de acesso / Núcleo | Apenas LTE | Apenas NR | LTE com NR para dados apenas | NR com LTE para dados apenas |
|---|---|---|---|---|
| EPC | Opção 1 (4G) | Opção 6 (desconsiderada) | Opção 3 | Opção 8 (desconsiderada) |
| 5GC | Opção 5 | Opção 2 | Opção 7 | Opção 4 |

**Figura 3.6. Opções definidas durante a fase inicial da definição do sistema 5G.**

Na Figura 3.6, a opção 1 corresponde ao sistema 4G existente e especificada em lançamentos anteriores da 3GPP. As opções 6 e 8 foram desconsideradas, pois introduziriam muitas limitações ao NR (*New Radio*) de 5G, para fornecer compatibilidade retroativa com o EPC. Embora as opções 2, 3, 4, 5 e 7 tenham sido consideradas nos trabalhos de especificação, foi definido que as prioridades seriam as opções 2 e 3. Assim, o Lançamento 15 da 3GPP introduziu as arquiteturas Não-Autônoma (NSA – *Non-Stand Alone*), opção 3, e Autônoma (SA – *Stand-Alone*), opção 2.

A arquitetura NSA foi proposta em maio de 2018 e seis meses depois, 3GPP apresentou as especificações para a arquitetura SA. NSA é uma forma rápida para prover conectividade de alta vazão de dados, pois permite o aproveitamento dos ativos de rede

existentes, sem a necessidade de implantar um novo sistema completo de ponta-a-ponta para a rede 5G, *i.e.*, uma arquitetura de transição da 4G para 5G. Na NSA, somente a tecnologia de rádio precisa ser atualizada. No entanto, para melhor explorar a capacidade da 5G, visando novos serviços, uma nova arquitetura independente do sistema 4G existente faz-se necessária. SA é considerada a arquitetura 5G definitiva, incluindo as melhorias na transmissão de rádio, juntamente com núcleo 5G nativo em nuvem. As duas arquiteturas propostas no Lançamento 15 da 3GPP estão ilustradas na Figura 3.7 e são discutidas nas subseções seguintes.

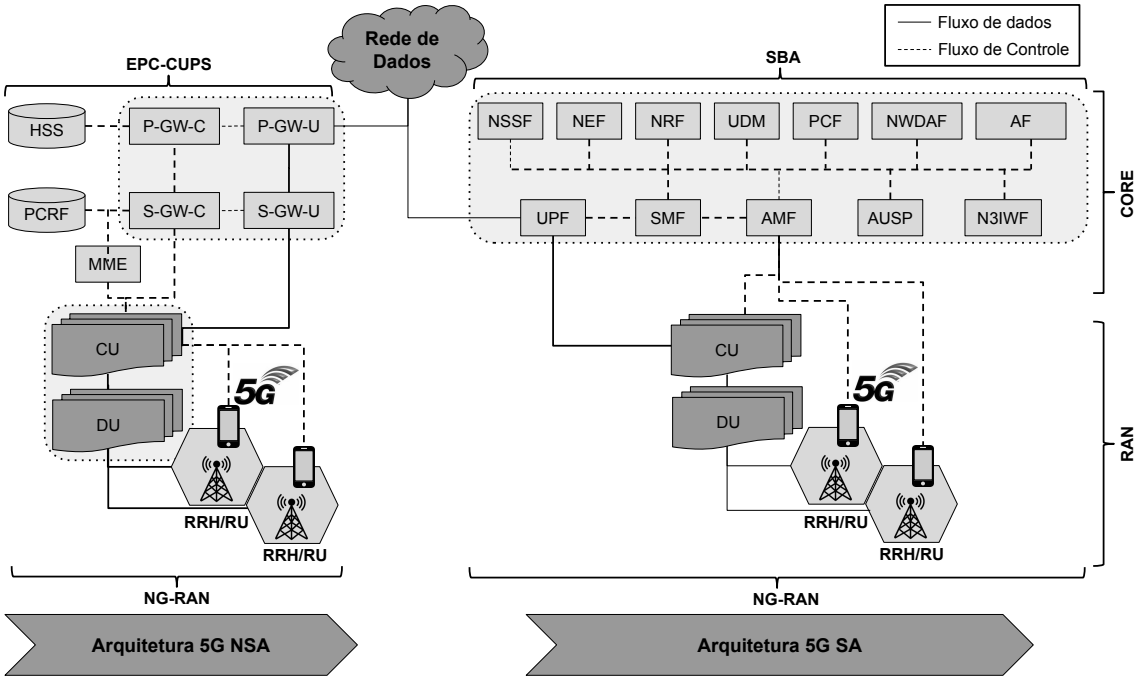



**Figura 3.7. Arquiteturas 5G: Não-Autônoma (à esquerda) e Autônoma (à direita).**

### 3.3.1. Arquitetura 5G Não-Autônoma (NSA - *Non-Stand Alone*)

A arquitetura NSA, também conhecida como EN-DC (E-UTRAN-NR *Dual Connectivity*), atende a filosofia da indústria de telecomunicações de introduzir novas tecnologias da maneira mais rápida e menos disruptiva. Embora as motivações sejam claras, é importante destacar algumas desvantagens da arquitetura NSA. Nessa arquitetura, a nova interface de rádio (NR) pode ser implantada apenas onde já existir cobertura 4G/LTE, indicando a ausência de autonomia da arquitetura, conforme sugerido pelo nome NSA. Além disso, as funcionalidades de rede disponíveis estão limitadas ao que é oferecido por LTE/EPC. Desta forma, várias novidades introduzidas no sistema 5G não estão disponíveis, tais como: fatiamento de rede, tratamento de QoS, flexibilidade na computação de borda e a extensibilidade geral do núcleo 5G, para incluir novas aplicações em um ambiente similar ao de nuvem tradicional.

O projeto da arquitetura NSA se focou principalmente em permitir que pelo menos duas inovações da 5G fossem introduzidas: (*i*) aumento sensível da capacidade de largura

de banda e vazão da rede, e (*ii*) maior flexibilidade nas funções do plano de usuário fornecidas pelos GWs (S-GW e P-GW) do núcleo EPC. Conforme ilustrado na Figura 3.8, a separação entre os planos de controle e dados nos GWs, e a conexão dupla através de LTE e NR são duas características importantes introduzidas na arquitetura NSA. Basicamente, foram introduzidos três conjuntos de tecnologias para implementar essas características: (*i*) *DEdicated* CORE *networks* (DECOR) e *enhanced* DECOR), (*ii*) *Control and User Plane Separation* (CUPS) e (*iii*) NR como RAT (*Radio Access Technology*) secundária. A seguir, descrevemos brevemente essas tecnologias.

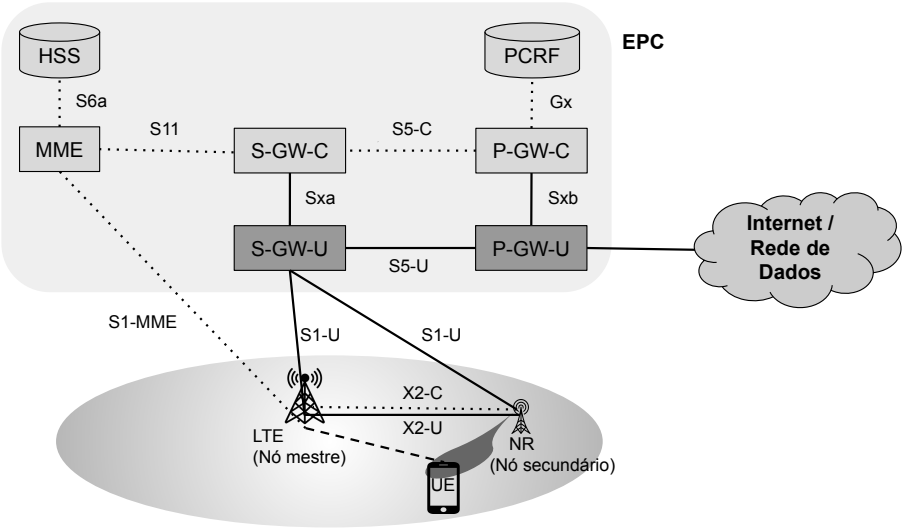
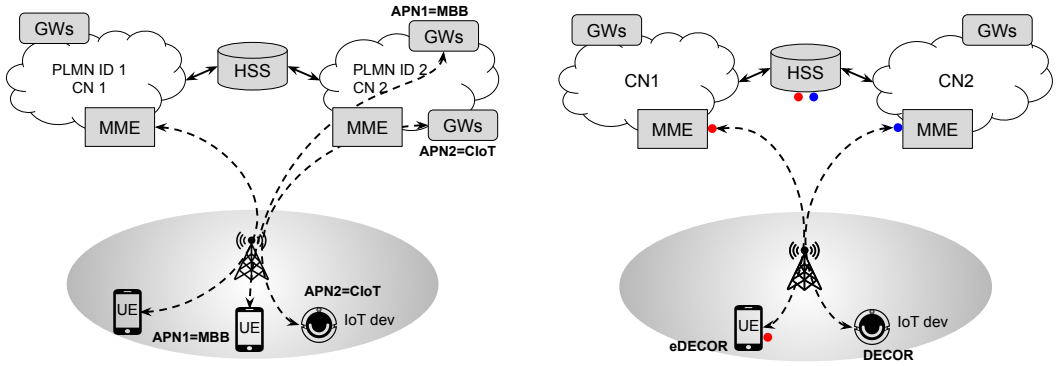



**Figura 3.8. Arquitetura NSA, com destaque para a separação entre controle e dados nos GWs e a conexão dupla (LTE e NR).**

### *DEdicated* CORE *networks* (DECOR) e *enhanced* DECOR)

Um operador de rede é reconhecido pelo seu identificador PLMN (*Public Land Mobile Network*), que originalmente corresponde a um núcleo de rede móvel. No entanto, ainda em 4G, houve uma demanda significativa por flexibilizar essa abordagem e permitir que um único operador pudesse instanciar múltiplos núcleos e também direcionar cada usuário para o núcleo adequado de acordo com o serviço necessário. Antes da introdução de eDECOR, a separação de núcleos era possível apenas utilizando diferentes identificadores de PLMN, ou seja, instanciando redes de núcleo independentes. Uma solução alternativa era utilizar APNs (*Access Point Names*) diferentes para direcionar os usuários para várias redes de serviço, que estavam associados a diferentes entidades do plano de usuário, em especial, P-GWs. Ambas as abordagens, ilustradas na Figura 3.9(a), eram pouco flexíveis.

Através da introdução de eDECOR, um operador pode implantar múltiplas redes de núcleo dedicadas (DCNs – *Dedicated Core Networks*) dentro de uma PLMN com cada DCN consistindo de um ou múltiplos nós da rede de núcleo (*e.g.*, apenas MME, MME com GWs, MME, GWs e PCRF). Cada DCN pode ser dedicada a servir um tipo diferente de UE, separando certos tipos de tráfego em nós da rede de núcleo específicos e, se necessários, os ajustando diferentemente do restante dos nós da rede de núcleo.




(a) pre-DECOR (b) DECOR e eDECOR

**Figura 3.9. Exemplo de uma configuração pre-DECOR, usando APNs, em comparação com o uso de DECOR e eDECOR.**

Com DECOR, as informações necessárias para identificar como encaminhar o tráfego do usuário são obtidas pela MME apenas no HSS, enquanto que eDECOR exige que o UE forneça informações específica (*i.e.*, a DCN desejada) para facilitar a seleção rápida e ótima da rede de núcleo dedicada. A Figura 3.9(b) ilustra o uso de DECOR (com informações inicialmente apenas no HSS, representadas pelo círculo azul) e eDECOR (com informações também no UE, representadas pelos círculos vermelhos) para acessar diferentes DCNs.

De fato, eDECOR pode ser considerada precursora da implantação do conceito de fatiamento de rede, já introduzindo algumas características similares como diferenciação de QoS e instanciação de múltiplos nós da rede de núcleo para atender cada serviço. Por outro lado, fatiamento de rede é um conceito mais avançado que permite o operador da rede móvel realizar o fatiamento fim-a-fim, desde a parte do rádio frequência, passando por toda a rede de acesso e incluindo todo o núcleo. Fatiamento de rede deve permitir criar fatias de recursos que são isoladas parcial ou totalmente, física ou logicamente.

### *Control and User Plane Separation* (CUPS)

A funcionalidade de separação entre planos de controle e de usuário (CUPS) surgiu da necessidade de dimensionar independentemente as funções de plano de usuário e de controle da rede (de pacotes) do núcleo para o gerenciamento de sessão e os serviços de dados. Antes da introdução de CUPS, não era possível implantar componentes de GW apenas com função (de dados) de usuário ou dimensionar independentemente, de maneira padronizada, as partes de plano de controle e de plano de usuário. A necessidade dessa separação tornou-se bastante clara à medida que as operadoras começaram a considerar os impactos em sua rede de recursos internos, como IoT (de banda estreita), banda larga móvel (*Mobile Broadband* – MBB), e também o crescimento de serviços OTT (*Over-The-Top*) baseados na Internet, como vídeo em fluxo armazenado, compartilhamento de conteúdo e comunicação em mídia social. Conforme ilustrado na Figura 3.10, CUPS pode ser aplicada aos seguintes elementos do EPS: S-GW, P-GW e função de detecção de tráfego (TDF - *Traffic Detection Function*).

Como pode ser visto na Figura 3.10, o plano de controle e o plano de usuário

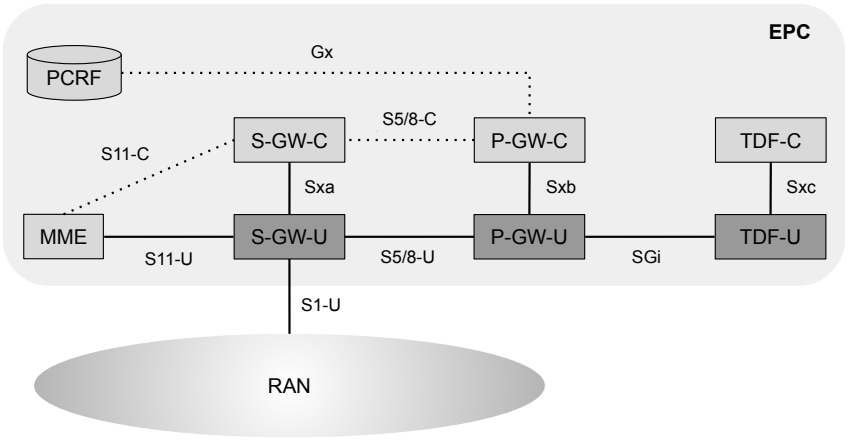



**Figura 3.10. Arquitetura básica do EPC com a separação entre os planos de controle e de usuário.**

de cada elemento (S-GW, P-GW e TDF) utiliza uma interface Sx (a, b e c, respectivamente) para realizar a comunicação necessária. A interface Sx oferece procedimentos para estabelecimento, modificação e encerramento, com intuito de fornecer suporte para as operações de plano de controle e plano de usuário entre os componentes de cada nó separado. A 3GPP definiu o PFCP (*Packet Forwarding Control Protocol*) para suporte a essas funcionalidades em Sx e esse protocolo também foi adotado em sistemas 5G.

Utilizando S-GW e P-GW como exemplos, é interessante destacar algumas características de projeto da CUPS. Desde o início, ficou evidente que nem todos os cenários exigiriam a separação entre os planos (de controle e de usuário) e que os cenários mais comuns de implantação teriam que contemplar a coexistência de nós com e sem a separação em uma única rede. Assim, foi definido que a separação entre os planos não deveria ter nenhum impacto em outros componentes do núcleo, como MME, PCRF, sistema de cobrança/tarifação e sistema de gerenciamento de assinaturas. Além disso, CUPS também foi projetada para não impactar nenhum procedimento ou protocolo que envolvesse UEs e elementos da RAN. Como consequência, outros elementos da rede não têm conhecimento se o S-GW e o P-GW com os quais interagem possuem ou não planos de controle e de usuário separados.

### NR como tecnologia de rádio de acesso secundária

A conectividade dupla (DC - *Dual Connectivity*) da arquitetura NSA permite que o UE e a RAN possam receber e transmitir dados por duas estações-base simultaneamente. Desta forma, a DC provê a capacidade de utilizar recursos de rádio fornecidos por dois grupos de células operando independentemente, mas conectados a um único núcleo. No contexto NSA, a principal motivação da conectividade dupla é aumentar a taxa de transferência do usuário, mas também é possível fornecer maior robustez de mobilidade e oferecer suporte a balanceamento de carga entre os nós da RAN. Inicialmente, o conceito de conectividade dupla foi introduzido para o EPS com dois grupos de células, ambos fornecendo recursos E-UTRAN. A BS que o UE se conecta primeiro (e a partir da qual toda a sinalização em direção ao núcleo é realizada) é conhecida como nó mestre e possui

a localização onde UE está associada. Quando o UE atingir o estado conectado, o nó mestre pode solicitar outra BS (*i.e.*, nó secundário) para descarregar o tráfego de dados. Posteriormente, a solução evoluiu para suportar múltiplas tecnologias de rádio (MR-DC - *Multi-Radio* DC), sendo EN-DC a variante que combina as tecnologias E-UTRAN e NR.

O requisito de flexibilidade do espectro foi um fator determinante para a adoção de tecnologias baseadas em OFDM em LTE para 4G e continua a ser um fator importante para o planejamento e as implantações de NR para 5G. Ao longo de 3G e 4G surgiu da necessidade de alocações em diversas bandas de frequência do espectro, diferentes larguras de banda de operação, múltiplos esquemas duplexação e esquemas de acesso múltiplo. No entanto, a largura e diversidade do espectro utilizado no NR representam uma das caraterísticas mais peculiares da 5G. O NR suporta um espectro grande e diversificado de 410 MHz a 52,6 GHz (e até 100 GHz em lançamentos futuros). Além disso, NR pode utilizar larguras de banda de 5 MHz a 3,2 GHz, suportando TDD (*Time Division Duplex*) e FDD (*Frequency Division Duplex*), assim como portadoras suplementares para enlaces de descida (SDL *supplementary downlink*) ou subida (SUL - *supplementary uplink*). Nesse contexto, a 3GPP definiu no Lançamento 15 duas faixas de frequência (*Frequency Range*): FR1 (410 MHz – 7125 MHz) e FR2 (24250 MHz – 52600 MHz). A FR2, também conhecida como faixa de ondas milimétricas, é a que oferece a maior capacidade, porém possui características físicas distintas da FR1, além de apresentar o maior desafio para implantação e uso em 5G. A seguir, apresentamos algumas informações adicionais sobre ondas milimétricas, as quais ainda são um assunto de significativo interesse acadêmico [31, 32].

A faixa de ondas milimétricas pode proporcionar comunicações com taxas na ordem de Gigabits e, por isso, é bastante atrativa para redes móveis. Entretanto, essas ondas impõem grandes desafios no projeto, implementação e operação dos sistemas, pois sofrerem maior degradação e atenuação na propagação, além de ser mais suscetíveis a bloqueios físicos e absorção atmosférica. Por outro lado, com o desenvolvimento de elementos de antenas cada vez menores, é possível habilita a construção de conjuntos de antenas para o emprego de MIMO massivo e formação de feixe (*beamforming*), que auxilia no tratamento dos problemas de propagação e melhora sensivelmente o reuso de frequência. No entanto, a formação/manutenção de feixe consome um tempo não negligenciável e seu aperfeiçoamento tem sido tema de vários trabalhos [33, 34]. Além disso, o problema se torna notadamente difícil quando é considerada a mobilidade dos usuários. Uma das estratégias da 3GPP para minimizar essas limitações é utilizar a arquitetura 5G Autônoma, como será apresentada na próxima subseção.

### 3.3.2. Arquitetura 5G Autônoma (SA - *Stand Alone*)

Na arquitetura SA, o núcleo EPC é substituído pelo núcleo SBA (apresentado detalhadamente na Seção 3.5), permitindo a utilização de um conjunto de funções de redes virtuais que fornecem serviços para outras funções da arquitetura ou, até mesmo, para aplicações de usuários finais. Desta forma, a arquitetura SA consolida o conceito de dissociação de dados e controle, permitindo o posicionamento flexível e sem estado de funções virtuais nos diferentes segmentos de rede que compõem um sistema 5G. Por exemplo, esses segmentos podem ser vistos como *edge*, *fog* e nuvem, ou ainda, especificamente na RAN, podemos encontrar segmentos como *fronthaul*, *midhaul*, *backhaul*.

Um grande avanço da arquitetura SA é seu direcionamento para o desenvolvimento de funções de redes virtuais baseado no modelo nativo da nuvem, i.e., (*Cloud Native*). Essa abordagem refere-se a como as funções de redes virtuais são criadas e implantadas de maneira flexível, utilizando amplamente o conceito de computação em nuvem para desenvolver, implantar e gerenciar serviços. Por exemplo, dentro dessa abordagem, o conceito de microsserviços pode ser utilizado para a implantação e expansão de um sistema 5G, alinhado as evoluções ágeis da tecnologia da informação. Um dos objetivos de utilizar microsserviços é poder decompor os componentes em funções especializadas, com baixa granularidade, para tornar o serviço leve e com alta capacidade de compartilhamento. Esse objetivo encaixa-se perfeitamente na necessidade de suportar os cenários de comunicação eMBB, mMTC e URLLC, uma vez que oferece modularidade, reutilização de funções e interoperabilidade com redes heterogêneas. Além disso, a introdução de microsserviços e interfaces para acessá-los simplifica os processos de atualização e manutenção de sistemas 5G, reduzindo o custo da operação e acelerando a introdução de novos serviços.

A implantação da arquitetura SA de forma plena apresenta alguns desafios. Por exemplo, historicamente as gerações de redes celulares são fortemente baseadas no contexto de sessões dos UEs, caracterizando uma arquitetura baseada em estado. Por outro lado, sistemas baseados no modelo de computação em nuvem nativa, possuem como característica fundamental a arquitetura sem estado, onde os contextos de seus estados são salvos em banco de dados. Por exemplo, o núcleo SBA introduz a função de armazenamento de dados não estruturada (UDFS - *Unstructured Data Storage Function*) para controlar o contexto das funções de redes, como pode ser observado na Figura 3.11. Essa característica permite usufruir de benefícios como elasticidade e gerenciamento dinâmico dos microsserviços em redes móveis. Um dos grandes desafios da 5G é projetar e desenvolver componentes de software baseados originalmente em estados, através da pilha de protocolos especificados pela 3GPP, para microsserviços sem estados, usufruindo plenamente dos benefícios da computação em nuvem nativa.

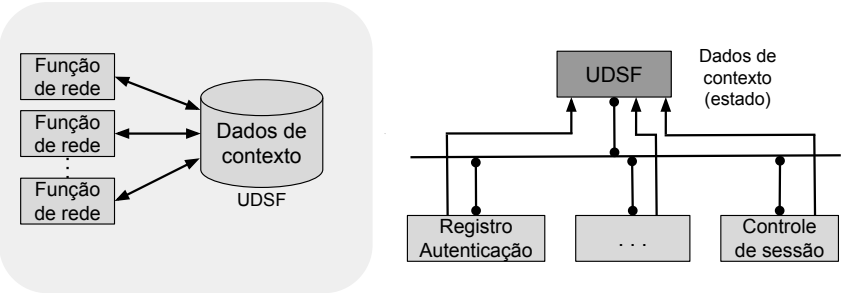



**Figura 3.11. Função de dados de contexto da arquitetura SA.**

Na arquitetura SA, a característica fundamental é que os elementos são definidos como funções de rede, as quais fornecem serviços a outras funções, ou qualquer outro *consumidor* autorizado, mediante interfaces de programação. Desta forma, a arquitetura fornece ampla modularização e reusabilidade em um completo isolamento entre o plano de dados e o plano de controle. Devido a essa alta modularização, a eventual migração

da arquitetura NSA para SA deve ser imperceptível para o usuário final. Além disso, vale ressaltar que a NSA e a SA não são propostas que competem, mas sim um caminho evolutivo para adoção das inovações introduzidas por 5G. A intenção é começar com NSA e migrar gradualmente para SA ao longo do tempo, em especial para operadores que já tenham investimento expressivo em redes 4G. Por algum tempo, NSA e SA devem coexistir e várias abordagens são possíveis. A Figura 3.12 ilustra uma solução potencial proposta pela Samsung [35], na qual surge uma arquitetura intermediária, utilizando um núcleo comum.

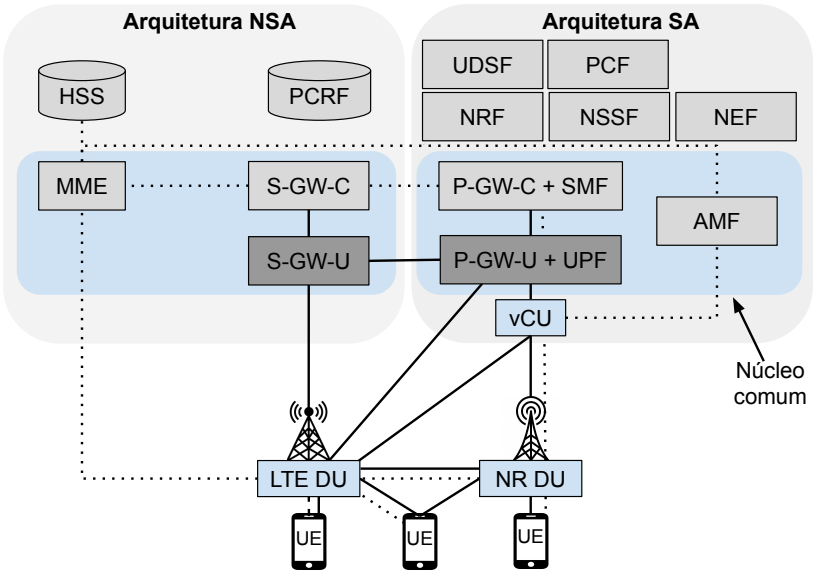



**Figura 3.12. Proposta para coexistência das arquiteturas NSA e SA.**

### 3.3.3. Pós-5G

Embora muitos países, como o Brasil, ainda estejam discutindo e se preparando para a implantação das redes 5G, 2019 é o ano em que a adoção efetiva começou em vários países. Alemanha, China, Coreia do Sul, Estados Unidos, Finlândia e Reino Unido são exemplos de países que já estão implantando, testando e/ou efetivamente utilizando redes 5G. Como esperado, a comunidade científica já começou a investigar questões sobre os limites das redes 5G, sobre aplicações (novas ou já existentes) que se beneficiariam de comunicação sem fio, mas não seriam devidamente atendidas por essas redes, sobre tecnologias que precisam evoluir ou serem criadas para atender novas potenciais demandas, dentre vários outros temas [36, 37]. Embora não seja possível prever precisamente como será a próxima geração de redes móveis sem fio e quais aplicações justificarão sua adoção, alguns temas são recorrentes nas discussões e investigações iniciais, sendo improvável que não sejam parte da evolução das redes 5G. Dentre esses temas, estão a alocação eficiente de recursos diversos em redes sem fio (*e.g.*, bandas sub-6GHz, ondas milimétricas e, futuramente, bandas em THz [38]), amplo uso de aprendizado de máquina e inteligência artificial, e aplicações que não são atendidas por redes 5G, tais como realidade estendida multissensorial e interações cérebro-computador sem fio.

Redes 5G tornaram notória a necessidade de considerar múltiplos tipos de recursos sem fio, além das tradicionais bandas licenciadas sub-3GHz. Em redes 5G, é possível utilizar a banda de 3,5GHz, banda não-licenciada de 5GHz, ondas milimétricas e comunicações D2D (*Device-to-Device*). Adicionalmente, já há recursos *Multi-access Edge Computing* (MEC) que permitem melhorar a comunicação dos dispositivos através de várias estratégias como armazenamento em cache, descarregamento de processamento, uso de informações de contexto, alocação de largura de banda, dentre outras. No entanto, uma oferta confiável de alta banda usando ondas milimétricas (posteriormente da banda de THz) em cenários com mobilidade é um desafio que dificilmente será superado em larga escala nas redes 5G. Devido a alta direcionalidade dos feixes de ondas milimétricas, é necessário que o alinhamento entre as antenas de dispositivos móveis e de BSs seja refeito com alta regularidade. Atualmente, isso significa rupturas na comunicação, mas à medida que cresce o número de BSs (tanto de ondas milimétricas, quanto de sub-6GHz) é possível mitigar o problema através da alocação adequada dos recursos. No entanto, essa alocação é complexa, pois envolve múltiplos dispositivos, múltiplas BSs, recursos diversos e deve ser realizada de maneira dinâmica em curtas escalas de tempo para atender a mobilidade dos usuários [39]. Baseado nesse contexto, existe um grande volume de dados que precisa ser processado rapidamente para tomada de decisão a respeito da alocação de recursos.

Além de modelos de otimização, o contexto descrito também pode fazer amplo uso de técnicas de aprendizado de máquina e inteligência artificial. De fato, essas técnicas foram inicialmente exploradas na 4G através do conceito de redes auto-organizáveis (SON – *Self-Organizing Networks*) [40]. Em redes 5G, aprendizado de máquina e inteligência artificial devem ter um uso mais amplo, dado o grande volume de dados a ser manipulado e a maior complexidade da rede [36]. No entanto, uma adoção ampla é prevista apenas após a consolidação das redes 5G e na próxima geração das comunicações móveis [41]. Além do aumento no volume de dados e da complexidade da infraestrutura, as evoluções recentes nas técnicas de aprendizado de máquina e inteligência tem motivado sua utilização mais ampla. Entretanto, para que soluções baseadas em inteligência artificial sejam massivamente empregadas em redes de comunicação, é importante que elas sejam construídas sob a perspectiva Inteligência Artificial Explicável (XIA – *Explainable Artificial Intelligence*) [42]. Entender o comportamento de soluções baseadas em inteligência artificial é um requisito importante para sua adoção em infraestruturas de comunicação.

As redes 5G estão introduzindo novas tecnologias de comunicação para atender uma ampla gama de aplicações e serviços. Por exemplo, eMBB, URLLC e mMTC fazem parte do sistema 5G para oferecer suporte a aplicações como vídeo 4K e 8K, carros autônomos, realidade virtual e aumentada e Indústria 4.0. No entanto, já estão surgindo aplicações que não são devidamente atendidas por redes 5G, tais como realidade estendida multissensorial, telepresença holográfica, interações cérebro-computador sem fio e sistemas autônomos e robóticos conectados [36, 37]. Em geral, essas aplicações apresentam novos requisitos de QoS que podem motivar a introdução de novas classes de serviço como comunicação de baixa latência confiável de banda larga móvel (MBRLLC – *Mobile Broadband Reliable Low Latency Communication*) e comunicações maciças de baixa latência ultra confiáveis (mURLLC – *massive* URLLC). Por outro lado, para explorar

de maneira mais eficiente os recursos disponíveis, é importante investigar e caracterizar adequadamente a demanda das aplicações, levando em conta não apenas requisitos de QoS, mas também requisitos de qualidade de experiência dos usuários (QoE – *Quality of Experience*) e de qualidade de experiência física (QoPE – *Quality of Physical Experience*) [36, 43].

## 3.4. Rede de acesso por rádio

Uma RAN é responsável por gerenciar a interface aérea para manter uma grande quantidade de usuários conectados. Desta forma, a RAN necessita eficientemente gerenciar o espectro de Rádio Frequência (RF), realizando uma série de tarefas, tais como gerenciamento de recursos de rádio, controle de mobilidade da conexão, alocação dinâmica de recursos para os equipamentos dos usuários, compressão e segurança de camada física, gerenciamento de sessão e fluxo de qualidade de serviço, dentre outras funções.

Originalmente, em uma RAN tradicional, o *front-end* de RF e as funcionalidades de processamento de banda-base eram integradas dentro de uma BS. O módulo da antena estava localizado a poucos metros do módulo de rádio, conectado com um cabo coaxial e apresentando altas taxas de perdas de transmissão. Essa arquitetura foi popular nas redes móveis de 1G e 2G. Nos últimos anos, o *front-end* de RF foi desacoplado do módulo de processamento de banda-base, permitindo os operadores móveis substituírem esse módulo de acordo com a demanda dos usuários. Esse desacoplamento arquitetônico permitiu avanços de centralização e virtualização no módulo de processamento de banda-base e no *front-end* de RF. A seguir, a Subseção 3.4.1 apresenta a evolução do conceito de centralização da arquitetura de banda-base e a Subseção 3.4.2 apresenta a virtualização da RAN, que iniciou na 4G e continua evoluindo em redes 5G.

### 3.4.1. Centralização da RAN

A alta taxa de transmissão de dados entre o domínio digital de processamento de banda-base e o domínio analógico do *front-end* de RF com a antena exige um barramento de alta largura de banda que conecta esses dois domínios. Essa exigência limitou, por muitos anos, o projeto de BS a componentes de hardware especializados com um barramento de alto desempenho conectando esses dois domínios. Essa limitação foi superada com a utilização de fibra ótica nesse barramento, reduzindo a perda de dados e aumentando a distância entre os domínios conectados. Nas gerações 3G/4G, especialmente a partir do Lançamento 8 da 3GPP, o processamento da banda-base passou a ser implementado em *BaseBand Units* (BBUs), *i.e.*, um hardware dedicado e especializado que implementa uma RAT, enquanto a *Remote Radio Head* (RRH) integra o *front-end* de RF com a antena. Na Figura 3.13, é ilustrada a evolução da arquitetura RAN de 2G para 3G/4G.

Em 3G/4G, o *front-end* de RF e o processamento de banda-base são claramente separados, como pode ser visto no lado direito da Figura 3.13. A RRH, também chamada de Unidade de Rádio, possui uma interface de fibra ótica e realiza a conversão analógica/digital e vice-versa, amplificação da potência e filtragem do sinal. O processamento de banda-base, agora realizado na BBU, é isolado e independente da unidade de rádio. Essa arquitetura é considerada descentralizada, uma vez que a BBU realiza sua operação separadamente da unidade de rádio. Avanços recentes impulsionaram a largura de

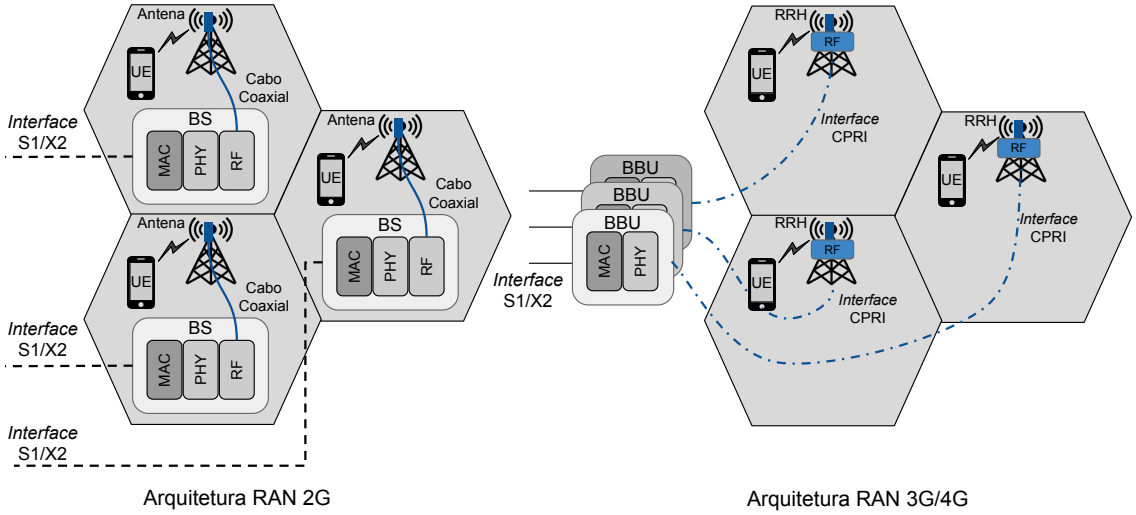



**Figura 3.13. Evolução das arquiteturas RAN.**

banda das fibras óticas permitindo incorporar arquiteturas baseadas em nuvem na rede de acesso por rádio (C-RAN – *Cloud Radio Access Network*). Nesse tipo de arquitetura, as BBUs podem estar localizadas em centro de dados (chamados arquiteturas de banda-base centralizadas), permitindo a redução de custo, através da manutenção centralizada e da elasticidade da computação em nuvem. Nessa nova configuração, as RRHs podem estar geograficamente separadas de um conjunto de BBUs por até aproximadamente 40 Km. Entretanto, é importante destacar que uma BBU é limitada ao processamento dos sinais de RRHs dentro de uma distância máxima, determinada de acordo com as restrições de atraso. Esse atraso é influenciado principalmente por três fatores: (*i*) distância entre BBU e RRH, (*ii*) condições do canal e (*iii*) capacidade de processamento disponível. Segundo Marotta et al. [44], a capacidade de processamento deve ser aumentada significativamente para RRHs que estão experimentando baixa Relação Sinal-Ruído (SNR – *Signal-to-Noise Ratio*) e estão a grande distância da BBU.

Os conceitos envolvidos em C-RAN têm chamado a atenção para a implementação inicial da 5G (Arquitetura NSA descrita na Subseção 3.3.1), principalmente considerando os benefícios de redução de custo e manutenção. Entretanto, BBUs são plataformas baseadas em hardware utilizando processadores de sinal digital especializados. Com objetivo de longo prazo, é fundamental substituir as BBUs baseadas em hardware especializados por software utilizando hardware de propósito geral, *i.e.*, BBUs virtuais (vBBUs). Da mesma forma, podemos pensar em uma camada de virtualização de rádio que permita várias tecnologias de acesso heterogêneas coexistindo sobre uma mesma RRH. Essa coexistência usa técnicas inovadoras de processamento de banda-base para dividir e abstrair uma RRH em várias RRH virtuais (vRRHs). Essa visão de virtualização da RAN está alinhada com a Arquitetura SA (descrita na Subseção 3.3.2) proposta no Lançamento 15 da 3GPP, onde por exemplo, a utilização de vBBUs é um caso de uso de NFV (*Network Functions Virtualization*) para prover serviços em um sistema 5G.

### 3.4.2. Virtualização da RAN

A virtualização da RAN (vRAN) tem recebido destaque em sistemas 5G, pois permite criar, gerenciar e configurar RANs dinamicamente, atendendo requisitos específicos de cada serviço. Além disso, o conceito de vRAN abre novos modelos de negócios em que provedores de serviços podem alugar vRANs dos provedores de infraestrutura. Nesse cenário, o provedor de infraestrutura controla todo o recurso físico, incluindo o espectro de RF, a RRH física, os recursos de hardware nos centros de dados (*i.e.*, servidores com processamento, memória e armazenamento) e a rede física. Um provedor de serviço pode contratar de um provedor de infraestrutura uma ou mais vRANs, incluindo ao menos uma BS virtual, *i.e.*, uma vRRH conectada a uma vBBU.

Isolamento, programabilidade e adaptabilidade são propriedades importantes para permitir a customização de vRAN para acomodar os diferentes serviços previstos para as redes 5G. Por exemplo, a combinação do fatiamento de BBU e RRH pode ser utilizada para instanciar uma vRAN fim-a-fim sobre uma infraestrutura física. Mais especificamente, o fatiamento e a virtualização podem ser implementados em uma BBU para permitir múltiplas vBBUs executarem sobre o mesmo hardware físico. Da mesma forma, uma RRH ou um conjunto delas podem suportar múltiplas vRRHs. Esses elementos aplicados para a vRAN, constituem os pilares para prover multi-serviços para as futuras redes móveis.

Um conjunto de vBBU e vRRHs pode ser executado em GPPs (*General Purpose Processor*), aproveitando as bibliotecas de processamento de sinal altamente otimizadas e aproveitando a evolução cada vez maior dos processadores, como maior poder de processamento e eficiência energética, como pode ser observado na Figura 3.14. Recentemente, o grupo de trabalho RAN3 da 3GPP [45] considerou a divisão de uma vBBU em duas novas entidades, denominadas *Data Unit* (DU) e *Control Unit* (CU). DU pode hospedar funções de tempo restrito da camada física, enquanto CU hospeda recursos de funções não críticas, como serviços de controle da camada MAC e superiores. A implementação da DU está prevista para cobrir uma área de 10 até 20 km de raio, enquanto a implementação da CU deve cobrir áreas de 100 a 200 km.

No Lançamento 15 da 3GPP manteve-se o conceito de nuvem na RAN, mas o nome foi alterado para próxima geração de redes de acesso (NG-RAN – *Next Generation Radio Acccess Network*) utilizando uma interface chamada NG. Além disso, devido à interoperabilidade entre 4G e 5G, DU e CU foram renomeadas para gNB-DU e gNB-CU. Uma gNB é responsável por uma série de tarefas, tais como gerenciamento de recursos de rádio, controle de mobilidade da conexão, alocação dinâmica de recursos aos equipamentos dos usuários, compressão e segurança de camada física, gerenciamento de sessão e fluxo de qualidade de serviço, dentre outras funções. Assim, a pilha de protocolos da gNB é detalhada considerando PHY, MAC, RLC, PDCP, RRC, etc.

A virtualização também introduziu a possibilidade de divisão da pilha de protocolos da gNB em funções de rede. Nesse contexto, 3GPP propôs oito opções para a divisão funcional entre as unidades centralizadas e distribuídas [45], considerando requisitos de transporte, em especial, vazão e latência. González-Día et al. [46] avaliaram, em um ambiente experimental, três opções de fatiamento referentes ao *fronthaul*, além de uma opção da unidade de dados do *backhaul*, como ilustrado na Figura 3.15. As oito opções de

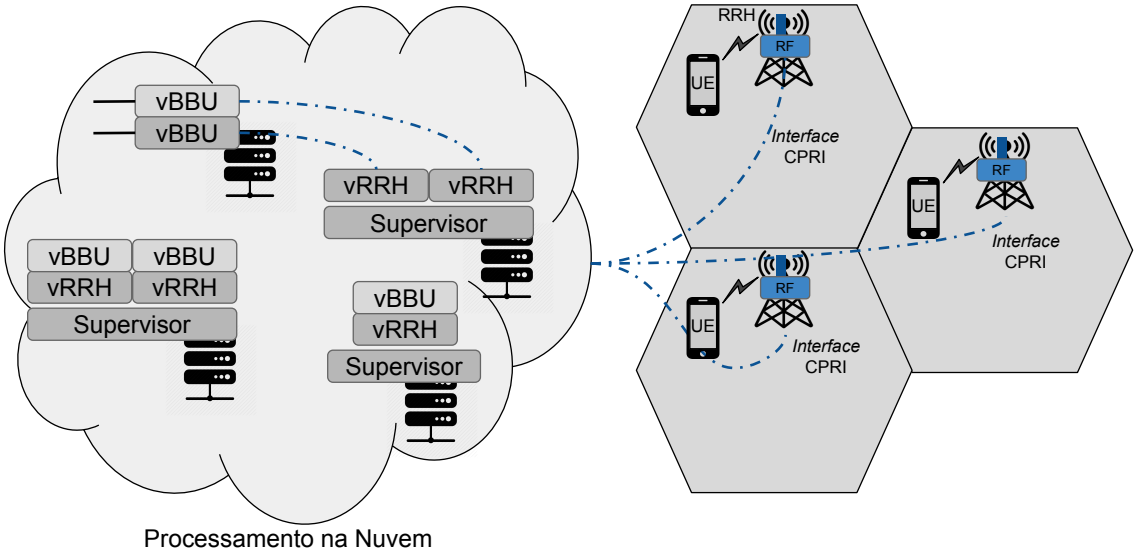



**Figura 3.14. RAN virtualizada.**

fatiamento propostas pela 3GPP ainda são tema de pesquisa [47] e desenvolvimento pela academia e indústria, pois requisitos de virtualização, processamento e funcionalidades ainda precisam ser investigados.

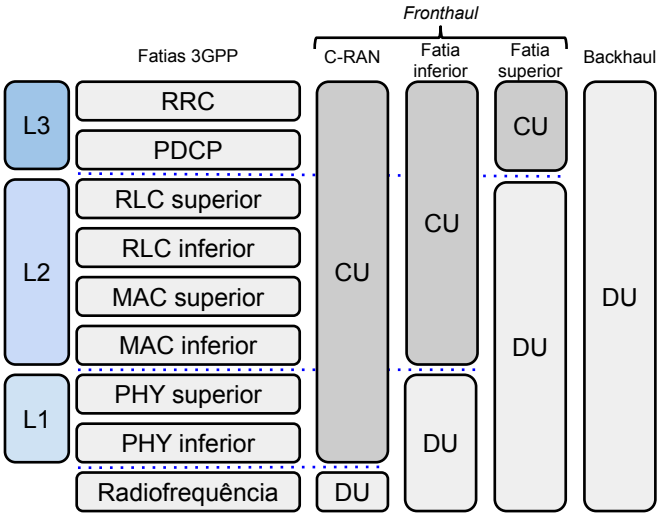



**Figura 3.15. Divisão de funções entre unidade central e distribuída.**

É importante ressaltar que essa nova arquitetura flexível da RAN composta de unidades distribuídas de dados (DU/gNB-DU) e controle (CU/gNB-CU) trouxe mudanças para a rede de transporte entre o acesso e o núcleo do sistema 5G. Atualmente, a rede de transporte está sendo reprojetada e desenvolvida considerando segmentos de *fronthaul*, *midhaul* e *backhaul*. O *fronthaul* é responsável pela comunicação entre a RRH/vRRH e o DU/gNB-DU. No segmento *midhaul*, ocorre a comunicação entre o DU/gNB-DU e

CU/gNB-CU. Por fim, o *backhaul* realiza a comunicação entre o CU/gNB-CU e o novo núcleo do sistema 5G. Também são comuns abordagens que adotam a integração entre os segmentos de transporte, em uma configuração de *crosshaul* [46, 47], na qual o objetivo explorar o uso eficiente de recursos de transporte de alto custo. orquestração das cargas de trabalho de vRANs nessa nova arquitetura é tema de grande investigação atualmente.

### 3.4.3. Demonstração de RAN

#### Objetivos

Um objetivo da demonstração é apresentar de maneira prática as funcionalidades de uma RAN, utilizando software e hardware abertos. Outro objetivo é comentar brevemente os processos de instalação e configuração do software que implementa a RAN, deixando material disponível para replicação desta demonstração.

#### Descrição

Nesta demonstração, utilizamos uma rede de acesso baseada em tecnologia LTE de projetos de código aberto. Além da RAN, é apresentado também software capaz de emular o funcionamento de UEs. A demonstração é organizada em 3 experimentos: (1) UE e RAN em software, sem núcleo; (2) UE, RAN e núcleo EPC em software; e (3) UE em hardware (celular convencional), RAN em hardware (SDR – *software-defined radio*) e software e núcleo EPC em software. Todos os componentes são implementados utilizando contêineres Docker que podem ser hospedados em uma infraestrutura de nuvem. A Figura 3.16 apresenta os componentes de software e hardware que são utilizados na demonstração da RAN.

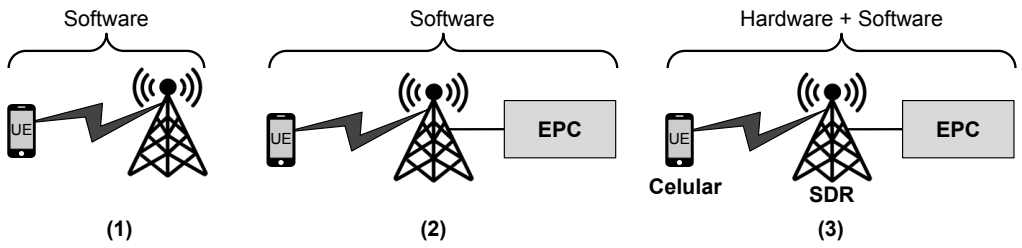



**Figura 3.16. Demonstração de uma RAN softwarizada.**

#### Mais informações

Durante o minicurso, são apresentados vídeos de demonstração dos experimentos. Além disso, estão disponíveis manuais com detalhes sobre como os experimentos podem ser replicados. Por fim, contêineres e qualquer código extra produzido pela equipe e que sejam necessários para replicar os experimentos estão também disponíveis publicamente.

Repositório deste minicurso:
`https://github.com/LABORA-INF-UFG/SBRC2020-Minicurso3`.

## 3.5. Núcleo da rede

Dentro do escopo de redes móveis, o núcleo pode ser caracterizado como o elemento de maior criticidade em um sistema 5G. O Lançamento 15 da 3GPP [6] estruturou o núcleo tendo como base um conjunto de componentes interconectados por uma camada de serviços. Cada componente possui uma responsabilidade específica, além de consumir e/ou fornecer serviços para os demais elementos do sistema 5G, através de APIs (*Application Programming Interface*) definidas pelo padrão. Essa estrutura, formada por componentes de software e suas interconexões via APIs, é ilustrada na Figura 3.17. O Lançamento 15 introduziu mudanças substanciais na maneira como as redes móveis são projetadas com o objetivo de fornecer suporte a uma ampla gama de serviços, cada um deles com requisitos de desempenho distintos [9].

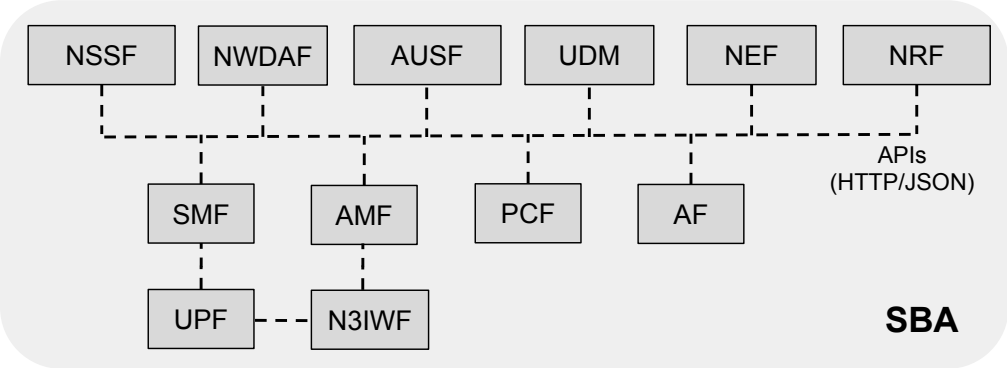



**Figura 3.17. Principais componentes de software do núcleo 5G.**

Dada a relevância do núcleo 5G, cada um dos elementos que o compõem necessitam ser robustos, resilientes, ter alta disponibilidade e escalabilidade. Essas são características normalmente associadas ao conceito de computação em nuvem. No contexto do Lançamento 15, cada um dos componentes que compõem o novo núcleo 5G podem ser decompostos em VNFs (*Virtual Network Functions*. Desta forma, cada VNF pode estar disponível sob demanda em uma infraestrutura na nuvem. A utilização baseada na decomposição de VNFs tornou-se possível porque os componentes não estão interconectados ponto-a-ponto através de interfaces físicas como em arquiteturas anteriores. Ao invés disso, cada um dos componentes do núcleo deve expor um conjunto de funcionalidades via software, através da arquitetura baseada em serviços (SBA) [48]. Nesse contexto, cada VNF oferece um ou mais serviços para outras VNFs. Pela nova definição da arquitetura SBA, a exposição de funções aplica-se apenas ao contexto de sinalização e não à transferência de dados do usuário.

Um dos métodos de comunicação que pode ser utilizado para uma arquitetura SBA tem como base REST (*Representational State Transfer*) sobre HTTP (*HyperText Transfer Protocol*). Esse método consiste em um conjunto de regras e diretrizes, amplamente utilizados na interconexão de sistemas distribuídos, que definem como as tecnologias de comunicação na Web acessam serviços através da utilização de APIs. Aparentemente, a expectativa por parte da 3GPP é tornar mais simples a tarefa de estender os recursos da rede. Outro aspecto a ser observado acerca do mecanismo de comunicação é que o

mesmo deve ser visto como uma opção lógica, dado que a especificação dos componentes que implementam as VNFs pressupõe que esses estejam executando em um ambiente virtualizado [49]. Os componentes que constituem o núcleo 5G podem ser vistos como uma malha interconectada de serviços. Cada VNF de um componente desempenha responsabilidades especificas e se interconectam a outras VNFs produzindo e consumindo serviços. As principais características de cada um desses componentes são descritas nas subseções a seguir. Cabe destacar que descrevemos brevemente os principais componentes presentes no núcleo 5G, sem o objetivo de listar todos os componentes e nem realizar uma apresentação exaustiva. A lista completa de componentes e sua descrição detalhada exige centenas (ou até milhares) de páginas do padrão ou um livro completo a respeito [50].

### 3.5.1. Principais componentes

#### *Access and Mobility Management Function* (AMF)

A mobilidade pode ser considerada a essência de um sistema 5G. O gerenciamento de mobilidade tem início no momento em que se estabelece uma nova conexão entre um UE e o núcleo da rede. Essa ação desencadeia uma série de procedimentos para identificar o UE, provendo uma estrutura de segurança, com o objetivo de fornecer um canal para o transporte de mensagens. O principal objetivo do componente AMF é garantir que o processo de comunicação se realize de forma coesa e transparente, considerando como fator chave a mobilidade do usuário. Através das funções implementadas no AMF a rede pode, por exemplo, alcançar um determinado usuário para notificá-lo sobre eventuais mensagens ou chamadas recebidas. Além disso, o componente AMF pode permitir, por exemplo, que um determinado UE dê início a um processo de comunicação com outros UEs também conectados na RAN ou que tenha acesso à Internet. Outra funcionalidade importante do AMF é garantir a conectividade e que as eventuais sessões existentes possam ser mantidas à medida que o UE se move entre diferentes pontos de acesso.

Em redes 5G, existe a necessidade de fornecer um suporte flexível para uma ampla gama de novos usuários [9]. Muitos desses usuários possuem necessidades específicas no que diz respeito a mobilidade. Por exemplo, um determinado UE utilizado em uma fábrica normalmente não se move, enquanto o UE em veículo autônomo ou controlado remotamente pode se apresentar alta mobilidade. Para melhor suportar essas diferentes necessidades, a especificação no Lançamento 15 [6] dividiu os procedimentos de mobilidade em três categorias para o componente AMF:

- Procedimentos Comuns podem ser caracterizados como um conjunto de etapas que serão executadas quando qualquer UE solicitar uma conexão com o núcleo. Dentre essas etapas, se destacam o processo de segurança, composta de autenticação primária, gerenciamento das chaves de acesso, identificação e configuração básica do UE.

- Procedimentos Específicos têm a função de gerenciar o registro e a atualização periódica da mobilidade de um determinado UE no AMF. Além disso, esse procedimento controla o encerramento do registro de um UE, dado um escopo de diferentes tecnologias de acesso.

- Procedimentos de Gerenciamento de Conexão são utilizados para estabelecer um processo de comunicação segura entre um determinado UE e o núcleo. Além disso, esse procedimento é utilizado quando um determinado UE necessita executar um processo de reserva de recursos da rede para o envio de dados.

Cada uma dessas categorias de procedimentos tem como objetivo prover funcionalidades que permitam UEs estabelecerem conexões com o núcleo da rede, utilizando os serviços associados à mobilidade.

***Session Management Function* (SMF)**

O componente SMF é responsável por gerenciar as sessões dos UEs, isto é, as sessões que representam os usuários conectados. As principais responsabilidades do SMF estão associadas às atividades de estabelecer, modificar e liberar as sessões individuais dos UEs, além de alocar os endereços IPs para cada UE conectado. Entretanto, a comunicação entre UEs e SMF é realizada de forma indireta através do componente AMF. Cabe ao AMF encaminhar as mensagens associadas à sessão de determinado UE e às funções do componente SMF.

As funções internas do componente SMF interagem com as demais VNFs dos outros componentes através do modelo produtor/consumidor, definida de acordo com a arquitetura SBA [48]. Por exemplo, o SMF desempenha a responsabilidade de controlar diferentes funções associadas ao componente UPF. Esse controle inclui a capacidade do SMF de configurar a direção do tráfego de dados associado a um UPF, para uma determinada sessão do UE. Além disso, o SMF deve executar ações de monitoramento e controle no UPF. O componente SMF também interage com funções associadas ao PCF, com o objetivo de executar a política das sessões dos UEs conectados. Essa ação de execução pode ser descrita como uma das principais tarefas associadas a um sistema 5G. Para ilustrar, é essa ação que determina as diretrizes para prover conectividade de dados entre um UE e a rede de dados (DN - *Data Network*).

***User Plane Function* (UPF)**

O Lançamento 15 da 3GPP [6] caracteriza o componente UPF como uma função fundamental dentro da nova arquitetura SBA. O UPF pode ser visto como parte do processo de separação entre Plano de Controle e Plano de Dados, inicialmente, introduzido no Lançamento 14 [51] com CUPS. O desacoplamento entre dados e controle, possibilita a arquitetura SBA descentralizar ainda mais seus componentes. Por exemplo, é possível direcionar atividades como processamento de pacotes para ficarem posicionados mais próximos da borda da rede, aumentando a QoS para o usuário e reduzindo o tráfego de rede.

As funcionalidades implementadas no componente UPF são controladas pelo SMF. A principal função do UPF está associada ao encaminhamento e ao processamento de dados dos UEs. Esse componente é ainda responsável por gerar notificações associadas ao tráfego de dados e pelo processo de inspeção de pacotes. O UPF funciona também como ponto estável de ancoragem entre o núcleo e as eventuais redes externas. O UPF possibilita que a comunicação aconteça de forma transparente, ocultando aspectos de complexidade associados à mobilidade. Pacotes IP destinados a um determinado UE são

encaminhados (da Internet) para o respectivo UPF, que esteja atendendo esse UE, mesmo quando o UE está em mobilidade. De uma forma geral, o componente UPF é responsável por:

- Desempenhar o papel de ancoragem entre o núcleo e as redes externas;
- Atuar como um ponto de acesso externo para PDUs (*Protocol Data Unit*), interconectando diferentes redes de dados;
- Realizar o roteamento/encaminhamento de pacotes, além de inspecionar os pacotes com o objetivo de detectar características de aplicações;
- Aplicar as definições associadas à gestão do plano de dados do usuário, bem como prover informações do tráfego de dados.

***Authentication Server Function* (AUSF)**

O AUSF é responsável pelo serviço que realiza autenticação dos UEs através das credenciais de acesso fornecidas pelo UDM. Além disso, o AUSF provê serviços associados a criptografia para possibilitar o tráfego de informações seguras e permitir a execução de processos de atualização de informações de deslocamento (*roaming*), bem como demais parâmetros associados ao UE.

De uma forma geral, os serviços providos pelo componente AUSF são consumidos pelas funções do AMF, as quais solicitam recursos associados ao processo de autenticação. As solicitações das funções do componente AMF são processadas internamente pelo AUSF e, posteriormente, delegadas a serviços providos pelo componente UDM, para execução de procedimentos de registro no repositório de dados.

***Unified Data Management* (UDM)**

O UDM é o componente responsável por gerenciar os dados dos usuários da rede em um único elemento centralizado. O UDM é equivalente ao HSS do núcleo EPC/4G. Através do UDM, diversas VNFs da arquitetura SBA conseguem executar diferentes ações, tais como: registro e autenticação de UEs, identificação de usuários, aplicação de políticas de acesso e autorização, dentre outras. O componente UDM interage diretamente com o AMF que encaminha as solicitações dos demais componentes. Além disso, em cenários onde existem mais de uma instância do componente AMF na rede, o UDM deve controlar quais instâncias serão as responsáveis por atender um UE específico [52]. Dentre as funcionalidades do UDM, se destacam [53]:

- Geração de credenciais de autenticação 3GPP AKA (*Authentication and Key Agreement*);
- Tratamento de identificação do usuário;
- Suporte à ocultação do identificador de assinatura protegido por privacidade;
- Autorização de acesso com base em dados de assinatura (por exemplo, restrições associadas à mobilidade);

- Gerenciamento de assinaturas;
- Gerenciamento de SMS.

O componente UDM funciona como a parte dianteira (*front-end*) para os dados de assinatura do usuário que são armazenados no UDR (*Unified Data Repository*). O UDM usa esses dados de assinatura para executar a lógica de diversas aplicações, como autorização de acesso, gerenciamento de registro e acessibilidade para finalização de eventos. O UDR é um banco de dados onde vários tipos de dados são armazenados e cujo acesso é oferecido como serviço para outros componentes como UDM, PCF e NEF. Existe ainda um componente de armazenamento opcional chamado UDSF (*Unstructured data storage function*), o qual permite que outros componentes (ou funções) armazenem dados de contexto dinâmico fora da própria função (ou componente). No contexto 3GPP, dados não estruturados se referem àqueles cuja estrutura não está definida nas especificações, permitindo que cada fornecedor que use um UDSF e escolha sua própria estrutura específica para o armazenamento. Não há exigência de qualquer compatibilidade de acesso ou armazenamento de dados em UDSF de diferentes fornecedores.

***Policy Control Function* (PCF)**

Esse componente desempenha a mesma função que o PCRF do EPC no sistema 4G. O PCF é o responsável por controlar o comportamento da rede, aplicando regras de segurança e controle, relacionadas ao gerenciamento das sessões, sobretudo àquelas funcionalidades associadas à mobilidade do usuário. Ele interage diretamente com o AMF, visando prover uma política de acesso e de mobilidade que pode incluir dentre outras coisas, o gerenciamento de restrições de acesso a serviços de uma determinada área, bem como a gestão de questões associadas à prioridade de acesso ao meio de determinados UEs em detrimento a outros (RFSP - *Radio Frequency Selection Priority*).

No contexto do gerenciamento de sessão, o PCF pode interagir com as funções de aplicação e com o SMF. O principal objetivo é prover métricas associadas à qualidade do serviço, bem como informações referentes ao fluxo de dados, as quais são obtidas pelo monitoramento regular de eventos associados à sessão de PDU. PCF também provê informações de políticas de segurança dos UEs. Essas políticas podem estar associadas a recursos de seleção de rede e regras para a seleção de fatiamento de recursos. O PCF pode ser acionado, por exemplo, para fornecer informações quando um determinado dispositivo realizar uma seleção de acesso (*UE access selection*) ou quando ocorrer o estabelecimento de uma sessão PDU [50]. A interação entre o PCF e as demais funções de aplicação, são implementadas através da exposição de seis serviços, a saber:

- *PCF-AM-PolicyControl* – provê informações relacionados a políticas de controle de acesso, seleção de rede, gerenciamento de mobilidade e diretrizes que podem ser aplicadas na seleção de rotas entre UEs e AMFs;
- *PCF-PolicyAuthorization* – fornece autorização e provê políticas de controle de acesso a uma requisição de um elemento AF, relacionado à sessão PDU a qual o AF está vinculado;

- *PCF-SM-PolicyControl* – fornece a um componente SMF, diretivas de acesso relacionadas à sessão PDU a qual o componente está vinculado;
- *PCF-BDT-PolicyControl* – fornece ao elemento NEF um conjunto de diretrizes que podem ser utilizadas por aplicações para realizar a transferência de dados em segundo plano;
- *PCF-UE-PolicyControl* – fornece diretrizes de controle que podem ser utilizadas no gerenciamento do processo de comunicação entre os UEs e demais funções de rede;
- *PCF-Event-Exposure* – possibilita que as demais funções de rede se inscrevam para serem notificadas quando um determinado evento vier a acontecer.

As decisões sobra a aplicação das políticas de monitoramento e controle realizadas pelo PCF são embasadas, em parte, por informações analíticas fornecidas por outras funções de rede, como a NWADF. O PCF também é um componente de fundamental importância em um cenário onde um AF necessite realizar uma atividade específica, por exemplo, a transferência de dados em segundo plano. Nesse caso, o AF pode entrar em contato com o PCF com o objetivo de inferir qual o melhor intervalo de tempo para a execução da atividade. Isso possibilita ao operador do sistema disponibilizar informações aos fornecedores de aplicativos sobre o momento mais adequado para se realizar a transferência de dados em segundo plano.

#### *Network Repository Function* (NRF)

O componente NRF é o repositório onde todas as funções disponíveis para uma determinada rede são listadas. O objetivo desse componente é permitir que outras VNFs consigam encontrar a função adequada para atender seus requisitos. O NRF tem a responsabilidade de selecionar o componente provedor de serviços mais adequado mediante critérios de desempenho fornecidos. Desta forma, o componente NRF é atualizado sempre que uma nova instância de VNF for implantada ou modificada. Além disso, o NRF detém informações sobre as demais VNFs, tais como tipo, capacidade, endereço, dentre outras.

Dentro do escopo SBA, o componente NRF desempenha papel fundamental para o funcionamento das demais VNFs. Esse componente oferece um mecanismo central que é capaz de automatizar todo o processo de configuração necessário para que as demais VNFs consigam descobrir e se conectar a outros serviços especializados.

### 3.5.2. Outros componentes importantes

#### *Network Slice Selection Function* (NSSF)

No contexto de redes 5G, o fatiamento de rede (*Network Slicing*) pode ser definido como a possibilidade do operador do sistema alocar um conjunto de recursos (em geral, virtualizados), com o objetivo de atender aos requisitos de determinados serviços ou aplicações [54]. O objetivo é fornecer suporte a um vasto conjunto de serviços, cada um deles com demandas específicas de desempenho [9]. As fatias de recursos virtuais são

na verdade instâncias lógicas de recursos de rede, os quais são necessários para atender a uma demanda específica. Uma fatia pode incluir recursos do núcleo e da RAN, e até extrapolar para redes de transporte entre diferentes núcleos e RANs, sendo considerada de serviço de rede do tipo ponta-a-ponta.

A arquitetura do núcleo 5G define o componente NSSF como sendo o responsável por gerenciar as instâncias de fatias de rede disponíveis. Ele é o responsável por selecionar as instâncias de fatias de rede, juntamente com o conjunto de AMFs disponíveis para um determinado UE. O AMF pode ser um componente dedicado a uma fatia específica ou atender a um conjunto de instâncias de fatias de rede. A função do NSSF é auxiliar o AMF na escolha das fatias de rede disponíveis, redirecionando do tráfego entre as fatias de rede controladas. O NSSF pode ser visto como um orquestrador, capaz de influenciar a maneira como o tráfego de rede é roteado. Ele produz dois serviços, um serviço seleção que produz informações acerca da fatia de rede selecionada e um outro serviço de disponibilidade que produz informações acerca das fatias de recursos disponíveis.

### *Network Exposure Function* (NEF)

O componente NEF é responsável por expor alguns eventos internos, relacionados a UEs e a arquitetura SBA. A exposição desses eventos visa atender a demanda de aplicações específicas e VNFs de outros componentes. Por exemplo, essas demandas necessitam ter acesso à localização de um determinado UE ou serem notificadas sobre a interrupção de conectividade de um determinado equipamento. Além disso, essas informações podem permitir o componente AMF ajustar o sistema de acordo com o comportamento de um grupo de usuários.

A possibilidade de expor eventos internos através de uma interface de acesso NEF abre novas oportunidades de negócio para provedores de serviços, permitindo que em alguns casos, serviços mais avançados possam ser oferecidos por terceiros. Por exemplo, uma aplicação pode utilizar os serviços expostos pelo componente NEF para saber se um determinado UE está ou não acessível, além de determinar a localização geográfica desse UE ou saber se o mesmo está em movimento. O componente NEF antede às solicitações das diferentes VNFs através de interações regulares entre os componentes UDM e AMF.

### *Network Data Analytics Function* (NWDAF)

O NWDAF é um componente opcional na arquitetura do núcleo e é o responsável por coletar diversos tipos de informações, oriundas da rede e de seus respectivos usuários. Essas informações são posteriormente organizadas e analisadas com o objetivo de prover os resultados inferidos para outras funções de rede. A descrição deste componente foi realizada de forma superficial no Lançamento 15 [6], sendo que sua especificação detalhada está prevista para o Lançamento 16.

Os dados coletados pelo NWDAF são oriundos de várias outras funções de rede que compõe o núcleo. A coleta dos dados é realizada através da camada de serviços que interliga os componentes do núcleo via serviço de escrita. Esse serviço pode ser acionado pelos eventos internos disparados por cada componente. O NWDAF também coleta informações sobre o funcionamento do sistema, bem como dados de registro de informações no componente UDR.

Segundo a especificação, os serviços fornecidos pelo NWDAF podem ser consumidos por qualquer outro componente do núcleo. O acesso externo também é possível através da utilização do mecanismo do componente NEF. As análises realizadas pelo NWDAF nos dados coletados ao longo do tempo podem ser utilizadas como um recurso histórico/estatístico visando prever valores futuros. Os dados analíticos produzidos pelo NWDAF podem também ser utilizados para aplicar determinadas ações no contexto da rede.

#### *Application Function* (AF)

AF é um componente genérico que representa uma possível aplicação, interna ou externa à rede da operadora, o qual interage com a arquitetura SBA. O processo de interação de AFs com SBA pode influenciar alguns aspectos de todo o sistema. Por exemplo, um AF pode interagir com o componente PCF através de serviços expostos pelo componente NEF, influenciando aspectos de QoS e, consequentemente, as políticas de cobrança/tarifação.

Um fator importante que deve ser avaliado pelo operador do sistema é o grau de confiança que um AF pode ter para interagir diretamente com determinadas VNFs. Por exemplo, um AF com maior confiabilidade poderia acessar diretamente VNFs de todos os componentes da arquitetura SBA, enquanto um AF menos confiável deveria primeiramente interagir com o componente NEF antes de ter acesso a funções mais sensíveis da rede.

#### *Non-3GPP InterWorking Function* (N3IWF)

O componente N3IWF é usado para integrar acessos não-3GPP com o núcleo 5G. O WiFi (IEEE 802.11) e DOCSIS (*Data Over Cable Service Interface Specification*) são exemplos de tecnologias de acesso não-3GPP previstas para integração pelo padrão. Conforme descrito anteriormente, o acesso 3GPP convencional utiliza uma BS, por exemplo, eNB (4G) ou gNB (5G). No entanto, o acesso não-3GPP se inicia em um dispositivo diferente, por exemplo, um ponto de acesso WiFi ou um modem HFC (*Hybrid fiber-coaxial*). Esse dispositivo usa o componente N3IWF para ter acesso à rede 3GPP, inclusive aos demais componentes do núcleo 5G.

Todo o tráfego a partir do componente N3IWF é transmitido através de canais seguros e é isolado de todos os demais tráfegos 3GPP. O isolamento é mantido não apenas para tráfego de dados (como é comum para comunicações 3GPP), mas também para tráfego de controle, inclusive o que ocorre antes do processo de autenticação. Mais detalhes do componente N3IWF, com sua utilização e a comunicação com outros componentes da arquitetura, são descritos na Subseção 3.6.1.

### 3.5.3. Demonstração de núcleo 5G

#### Objetivos

O objetivo da demonstração é apresentar de maneira prática as principais funcionalidades do núcleo 5G baseado em SBA. Além disso, comentaremos os processos de instalação e configuração do software que implanta o núcleo 5G, deixando disponível material para replicação desta demonstração. Pretendemos também apresentar alguns co-

mentários sobre o desenvolvimento de um núcleo 5G, mostrando como a estrutura inicial do código pode ser criada a partir das especificações da 3GPP.

### Descrição

Nesta demonstração, utilizamos o núcleo 5G baseado na arquitetura SBA de código aberto. Além do núcleo, é apresentado também software capaz de emular o funcionamento de RAN e UEs. A demonstração é organizada em 2 experimentos: (1) UEs, RAN e núcleo em software e (2) UE em hardware (celular convencional), UEs emulados por software, RAN em hardware (SDR), RAN emulada por software, bem como o núcleo 5G em software. Todos os componentes são implementados utilizando contêineres Docker que podem ser hospedados em uma infraestrutura de nuvem. A Figura 3.18 apresenta os componentes de software e hardware que são utilizados na demonstração do núcleo 5G.

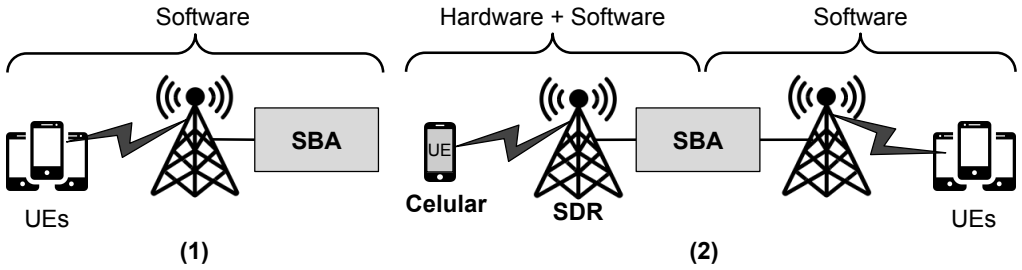



**Figura 3.18. Demonstração do núcleo 5G.**

### Mais informações

Durante o minicurso, são apresentados vídeos de demonstração dos experimentos. Além disso, estão disponíveis manuais com detalhes sobre como os experimentos podem ser replicados. Por fim, contêineres e qualquer código extra produzido pela equipe e que sejam necessários para replicar os experimentos estão também disponíveis publicamente.

Repositório deste minicurso:
`https://github.com/LABORA-INF-UFG/SBRC2020-Minicurso3`.

## 3.6. Redes 5G integradas a redes de acesso sem fio heterogêneas

Embora as redes 5G tenham sido projetadas para atender uma ampla variedade de serviços, oferecendo uma vasta gama de recursos, a 3GPP previu no Lançamento 15 a integração e suporte a outras tecnologias de acesso sem fio não-3GPP, com atenção especial para WiFi (IEEE 802.11). Como já foi discutido em outros trabalhos [55], [23], a coexistência de diferentes tecnologias de acesso promovem ganhos significativos em relação a desempenho e custo da comunicação de dispositivos de IoT. Além disso, existe uma imensa quantidade de dispositivos que possuem interfaces sem fio não-3GPP e que estarão em uso durante alguns anos. A integração de múltiplas tecnologias de acesso sem fio em um único núcleo tem o potencial de beneficiar tanto usuários, quanto administradores de infraestrutura de comunicação pública e privada.

O Lançamento 15 [56] já define como uma tecnologia de rede sem fio não-3GPP pode ser usada como acesso e integrada a um núcleo 5G. No entanto, esse lançamento visa apenas redes locais sem fio (WLANs – *Wireless Local Area Networks*), *i.e.*, redes WiFi, considerando o acesso não-confiável (*untrusted*). O acesso confiável (*trusted*) será introduzido no Lançamento 16 [57]. Nesse lançamento, está sendo dada atenção também às tecnologias de acesso a cabo, em especial HFC. Embora os padrões 3GPP não descrevam de forma específica como integrar outras tecnologias de comunicação sem fio usadas em IoT, como LoRa ou ZigBee, o arcabouço utilizado permite extensões que podem ser exploradas para esse fim [58, 59]. A seguir, apresentamos mais detalhes sobre o Lançamento 15 e também sobre a tecnologia LoRa/LoRaWAN, que usaremos como ilustração da integração entre o núcleo 5G e uma tecnologia de acesso sem fio não-3GPP, ao fim desta seção.

### 3.6.1. Redes de acesso não-3GPP

A arquitetura do núcleo 5G prevê, desde seu projeto, a possibilidade de integrar tecnologias de acesso não-3GPP, tendo como premissa que as interfaces de comunicação oferecem conectividade IP, *i.e.*, uma pilha IP tradicional. Embora essa premissa seja atendida por tecnologias como WiFi e HFC, várias tecnologias IoT, como LoRa, ZigBee, nRF24 e outras assumem que existirá um gateway para prover a conectividade IP para os dispositivos IoT. Portanto, vamos assumir a partir deste ponto no texto que um equipamento com pilha IP tradicional, *i.e.*, um dispositivo ou um gateway, estará disponível para estabelecer a integração. É importante destacar, como descrito em [59], que existem diferentes maneiras de realizar a integração de dispositivos que dependem de um gateway com pilha IP tradicional.

O núcleo EPC da 4G também possui suporte para redes de acesso não-3GPP, porém, usando uma abordando mais complexa. Além de definir entre acesso confiável e não-confiável, é preciso escolher entre mobilidade baseada em rede e mobilidade baseada em hospedeiro. Dada o foco do minicurso, não serão apresentadas mais informações sobre a abordagem usada no núcleo EPC, mas uma descrição completa pode ser encontrada em [60]. Para o núcleo SBA da 5G, foram definidas menos opções, basicamente, acesso não-3GPP **não-confiável** (Lançamento 15, congelado em março de 2019 e concluído em junho de 2019) e **confiável** (Lançamento 16, com previsão de congelamento para março de 2020 e conclusão junho de 2020). Nesse contexto, o termo não-confiável significa que o operador da rede 3GPP não confia na segurança oferecida pela rede de acesso não-3GPP e, portanto, precisa tomar ações que garantam o transporte adequado do tráfego dessa rede de acesso. Isso significa que o tráfego não-3GPP deve ser isolado dos demais, inclusive no núcleo 5G, que é adequado para aplicações e serviços IoT. O acesso confiável não há prevê integração de outras tecnologias de comunicação sem fio usadas em IoT, como LoRa ou ZigBee.

Para dar suporte a redes de acesso não-3GPP não-confiável, o principal componente introduzido pelo Lançamento 15 da 3GPP foi o N3IWF (*Non-3GPP Interworking Function*), que é responsável por reencaminhar sinalização e dados entre o núcleo 5G e a rede de acesso não-3GPP, como descrito na Subseção 3.5. A Figura 3.19 ilustra a integração de duas redes de acesso não-3GPP a um núcleo 5G, mostrando os principais componentes envolvidos e suas interfaces de comunicação. O componente N3IWF seleci-

ona o AMF para servir o dispositivo (ou gateway) IoT, a qual será responsável por cuidar da (eventual) mobilidade do dispositivo assim como intermediar toda a sinalização com outras funções do núcleo 5G. Para se comunicar efetivamente, um dispositivo precisa de sessões PDU, que são estabelecidas, modificadas e liberadas sobre o controle do componente SMF. Esse componente funciona como um plano de controle que opera sobre plano de dados implementado através do UPF.

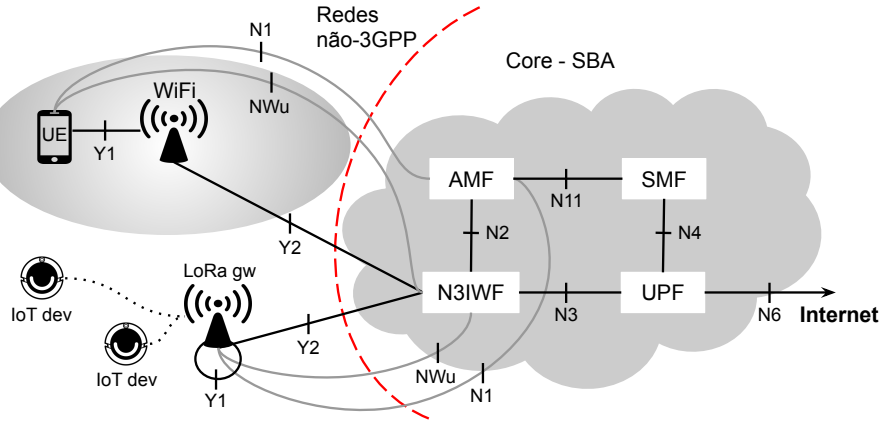



**Figura 3.19. Integração entre redes de acesso não-3GPP não-confiável ao núcleo SBA da 5G.**

No núcleo SBA da 5G, o componente AUSF permite que um dispositivo se autentique para ter acesso à rede e seus serviços, podendo usar tanto uma interface sem fio 3GPP (*e.g.*, NR) quanto uma interface não-3GPP (*e.g.*, WiFi). Os dispositivos não-3GPP podem autenticar com o núcleo SBA através de um esquema baseado em certificados, usando EAP-TLS (*Extensible Authentication Protocol - Transport Layer Security*) ou EAP-TTLS (EAP - *Tunneled* TLS), além do procedimento tradicional com credenciais baseadas em SIM (*Subscriber Identification Module*). A Figura 3.20 ilustra as conexões estabelecidas para fornecer a integração entre as redes de acesso não-3GPP não-confiável e o núcleo SBA da 5G.

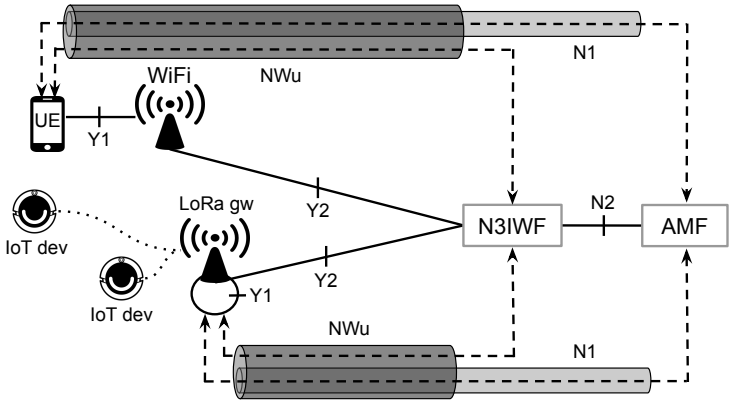



**Figura 3.20. Conexões para a integração de redes de acesso não-3GPP não-confiável com o núcleo SBA da 5G.**

A conexão Y1 não é especificada pela 3GPP, mas consiste em estabelecer a comunicação do dispositivo com sua rede de acesso, por exemplo, através de um ponto de acesso WiFi ou um gateway LoRa. É assumido que existe algum processo de autorização através da conexão Y1, que permite o dispositivo ter acesso à rede e obter um endereço IP. No entanto, várias tecnologias, como LoRa, não atribuem IP diretamente para os dispositivos e, portanto, parte de conexão Y1 precisa ser implementada no próprio gateway da tecnologia. A conexão Y2 também não é especificada pela 3GPP, mas deve oferecer a comunicação entre a rede de acesso e o componente N3IWF do núcleo SBA. Essa conexão pode ser estabelecida praticamente de qualquer forma, inclusive através da Internet pública, *i.e.*, potencialmente com múltiplos equipamentos e tecnologias intermediárias.

O padrão 3GPP especifica que entre o dispositivo (ou gateway) e o componente N3IWF deve ser estabelecido um túnel IPSec criptografado, chamado de NWu, para envio de tráfego de dados e de sinalização. o componente N3IWF seleciona o AMF para servir o dispositivo, sendo estabelecida a interface N2 entre as duas funções. Após estabelecimento da NWu e da N2, o padrão 3GPP especifica que é possível estabelecer a interface N1, entre dispositivo (ou gateway) e o componente AMF, para transportar sinalização NAS (*Non-Access Stratum*). Essa abordagem difere da utilizada no núcleo EPC da 4G, no qual a sinalização NAS se aplica apenas a redes de acesso 3GPP. Na SBA do núcleo 5G, dispositivos se conectando através de redes de acesso não-3GPP são gerenciados de maneira similar a dispositivos se conectando através de redes de acesso 3GPP.

### 3.6.2. LoRa

A tecnologia LoRa, desenvolvida pela Semtech Corporation [61], permite comunicação a longas distâncias utilizando o conceito de espalhamento espectral (*Spread Spectrum*) de frequências de rádio. Essa tecnologia apresenta os seguintes parâmetros de configuração que influenciam diretamente no desempenho da comunicação:

- Fator de Espalhamento (SF – *Spreading Factor*) com valores de 7, 8, 9 ou 10. Quanto maior o SF, mais informações são transmitidas por símbolo, gerando um ganho na transmissão;
- Largura de Banda (BW – *Bandwidth*) de 125 kHz, 250 kHz ou 500 kHz para um dado SF. Uma largura de banda mais estreita, aumenta a sensibilidade de recepção e incrementa o tempo de transmissão de um pacote;
- Taxa de Correção de Erro *(CR – Code Rate)* é responsável pela detecção e correção de erros.

Essas configurações determinam a taxa de bits transmitida, o tamanho máximo dos dados transmitidos e o tempo de duração da transmissão de um pacote no espectro de RF. Desta forma, essas configurações também influenciam o tamanho das mensagens, o seu alcance e o consumo de energia do dispositivo IoT.

Segundo [62] e [63], LoRa é resistente a interferência, devido à sua ampla gama de SFs, que podem ser empregadas em ambientes urbanos, rurais e até mesmo industriais. Além disso, essa tecnologia oferece cerca de quatro vezes mais alcance de cobertura, se

comparada com outras tecnologias de rádio, devido à sua natureza robusta e capacidade de atuar com sinais de rádio de baixa intensidade. Sua capacidade de realizar múltiplas transmissões no mesmo canal de RF, com a utilização de diferentes SFs diminui a probabilidade de colisões, aumentando a taxa efetiva de transmissão de dados, além de permitir discriminar entre erros de tempo e frequência. Desta forma, LoRa é considerada uma das tecnologias mais promissoras para a camada física de redes de sensores e, consequentemente, para a IoT [64]. Entretanto, para se ter uma aplicação efetiva de sensores e IoT é necessário ter uma rede, como LoRaWAN, que é descrita a seguir.

### 3.6.3. Rede LoRaWAN

Rede LoRaWAN é o nome dado as redes LPWAN (*Low Power Wide Area Network*) para IoT que utilizam como meio físico a tecnologia LoRa e implementam uma arquitetura baseada no protocolo de controle de acesso ao meio (MAC – *Media Access Control*) LoRaWAN. Na Figura 3.21, podemos observar a camada MAC, com suas subcamadas, enquanto a camada física LoRa habilita o enlace de comunicação de longo alcance. O protocolo da camada MAC e a arquitetura de rede têm grande influência na determinação da vida útil da bateria de um sensor, na capacidade da rede, na qualidade do serviço, na segurança e na variedade de aplicações IoT atendidas.

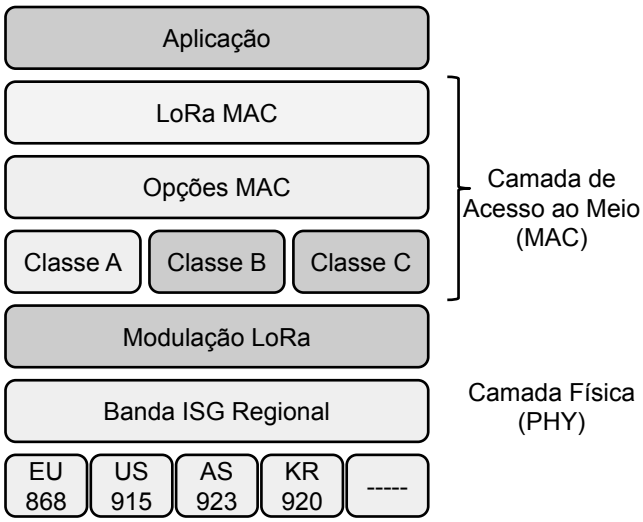



**Figura 3.21. Camadas da arquitetura de rede LoRaWAN.**

O protocolo LoRaWAN é baseado em padrões abertos e de baixo custo. Esse protocolo foi projetado desde o início para implementar plataformas para a IoT. Desta forma, o escopo de aplicação IoT suportadas é muito abrangente, tornando essa tecnologia muito atraente para ser conectada a sistemas 5G e criar uma rede IoT com grande área de cobertura. Entre as possíveis aplicações, encontram-se redes inteligentes para energia elétrica, rede de sensores de diversos tipos, agricultura de precisão, dentre outras.

A Figura 3.22 ilustra a topologia de referência do protocolo LoRaWAN. Uma característica proeminente dessa tecnologia é o baixo consumo, devido à topologia estrela que não necessita de roteamento entre nós. O elemento central é o gateway LoRaWAN

que pode ser conectado ao sistema 5G em um acesso não-3GPP não-confiável. Esses gateways, também conhecidos como concentradores, possuem modelos que podem atender a necessidades específicas, mais simples e baratos para ambientes fechados, em contraposição aos modelos industriais com proteção contra intempéries e aplicações externas. Na comunicação entre gateway e dispositivo, são previstos dois tipos de troca de mensagens: mensagens sem confirmação (*Unconfirmed Data Message*), semelhantes às mensagens UDP (*User Datagram Protocol*), e com confirmação (*Confirmed Data Message*), semelhantes ao TCP (*Transmission Control Protocol*). Outra característica importante da rede LoRaWAN é a existência de 3 classes de dispositivos que se comunicam com os gateways, os quais são descritos a seguir.

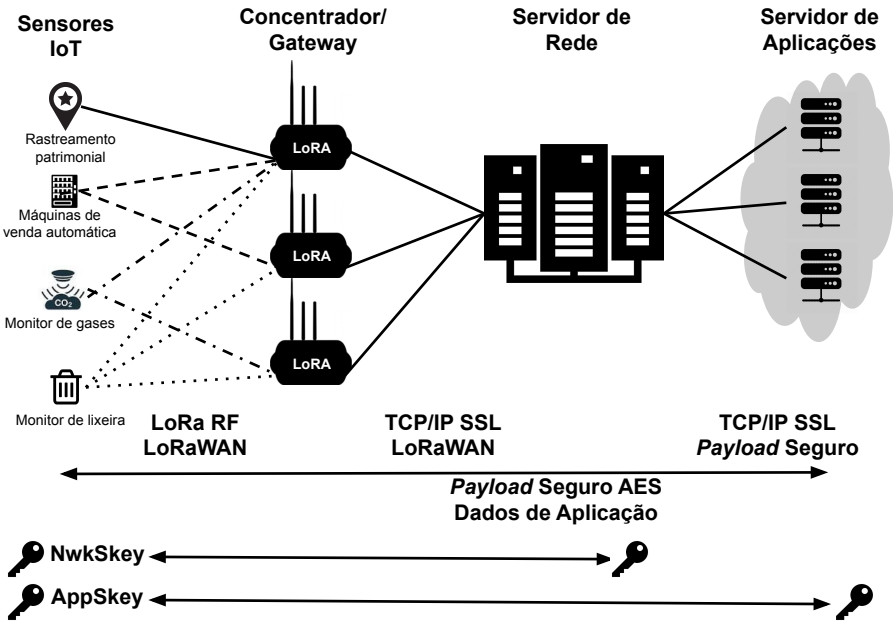



**Figura 3.22. Topologia de uma rede LoRaWAN**

- Dispositivos finais bidirecionais Classe A: as transmissões ascendentes (dispositivo-gateway) são baseadas no protocolo ALOHA. A recepção dos pacotes do gateway só pode ser realizada em duas curtas janelas de recepção que se abrem depois de uma transmissão. Essa classe fornece o menor consumo de energia, tendo como principal aplicação o monitoramento de grandezas.

- Dispositivos finais bidirecionais Classe B: além do modo classe A, são abertas janelas de transmissões (gateway-dispositivo) pré-programadas e gerenciadas por um sinal de temporização (*Beacon*), o qual indica quando o receptor está pronto para receber o sinal. Essa classe pode ser conveniente para sistemas de telecomando não sensíveis a tempo.

- Dispositivos finais bidirecionais Classe C: a janela de transmissão (gateway-dispositivo) está aberta continuamente, fechando-se apenas no momento da transmissão ascendente (dispositivo-gateway).

Em relação à ativação dos dispositivos na rede, a rede LoRaWAN adota o algoritmo de criptografia AES-128, utilizando duas formas diferenciadas de ativação em razão do tipo de rede Pública ou Privada. Nas redes públicas, utiliza-se o OTAA (*Over The Air Activation*), o qual se baseia no envio de um identificador global único (DevEUI), análogo ao endereço MAC, do identificador (AppEUI) e chave da aplicação (AppKey) desejada. Esses dados são utilizados na camada de aplicação para validar e ativar o dispositivo em determinada aplicação na rede. Ocorrendo a aceitação na rede, o dispositivo recebe uma mensagem de `join accept`, a qual contém o endereço do dispositivo (DevAddr), a chave de sessão da rede (NwkSKey) e a chave de sessão da aplicação (AppSKey). Nas redes privadas, são necessários os seguintes dados para ativação: endereço do dispositivo (DevAddr), chave de sessão da rede (NwkSKey) e chave de sessão da aplicação (AppSKey), os quais são gravados no dispositivo, no momento da sua configuração. Desta forma, o dispositivo está pronto para comunicação, quando estiver conectado na rede.

Na arquitetura de uma rede LoRaWAN, ainda estão presentes os Servidores de Rede, que são responsáveis pelo gerenciamento das informações enviadas pelos *gateways*. Como existe a possibilidade de dois ou mais *gateways* receberem o mesmo pacote de um determinado dispositivo, o servidor de rede elimina pacotes duplicados, gerencia os tempos para retorno de reconhecimento (ACK – *acknowledgement*) e faz os ajustes para adaptar as taxas de dados (DR – *Adaptative Data Rate*) de forma a gerenciar os tempos entre as comunicações e o consumo de energia. Finalmente, a rede LoRaWAN possui um ou vários Servidores de Aplicações que recebem via requisição ou de forma automática os pacotes dos Servidores de Rede e, de acordo com a requisição, executam uma ou mais ações específicas fornecendo a interface necessária para as diversas aplicações clientes.

### 3.6.4. Demonstração da integração de rede de acesso sem fio não-3GPP e o núcleo 5G

#### Objetivos

O objetivo da demonstração é apresentar a integração de uma rede de acesso sem fio não-3GPP ao núcleo 5G. Além disso, esse experimento mostra a grande capilaridade que um sistema 5G pode ter, incluindo redes com frequências não licenciadas.

#### Descrição

Nesta demonstração, combinamos uma rede de acesso baseada em tecnologia LTE com uma rede sem fio LoRa implementada em hardware. Para rede de acesso LTE usamos uma implementação do projeto de código aberto com SDR. O núcleo da rede 5G é baseado na implementação da SBA também fornecida em código aberto e implementada em software. Todos os componentes são implementados utilizando contêineres Docker que podem ser hospedados em uma infraestrutura de nuvem. Conectado à rede de acesso através do LTE, há um gateway LoRA implementado em hardware genérico com suporte a múltiplos sensores IoT que sincronizam seus dados com o servidor LoRa presente na outra extremidade da infraestrutura. Sendo assim, para fins de demonstração, os dados coletados pelos sensores são transmitidos através da rede LoRa até o gateway que encaminha os dados através da rede LTE, passando pelo núcleo SBA da rede 5G até alcançar o servidor LoRa. A Figura 3.23 apresenta o experimento que implementamos para a de-

monstração sobre a integração de uma rede de acesso sem fio não-3GPP com um núcleo de rede 5G.

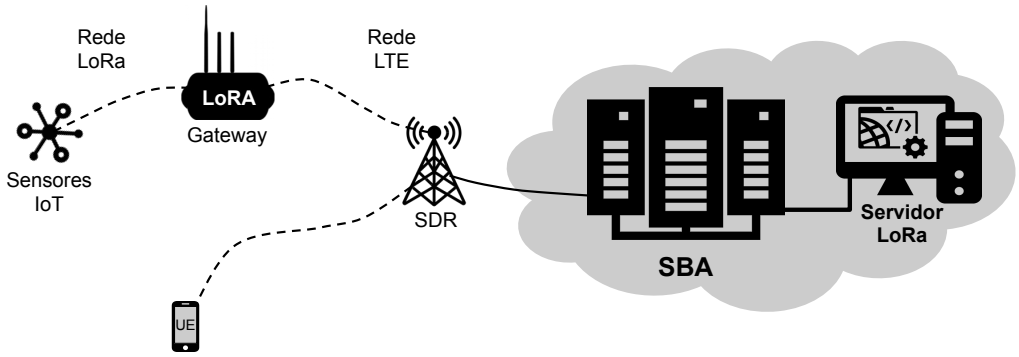



**Figura 3.23. Demonstração da integração de redes heterogêneas.**

### Mais informações

Durante o minicurso é apresentado um vídeo de demonstração do experimento. Além disso, está disponível um manual com detalhes sobre como o experimento pode ser replicado. Por fim, contêineres e qualquer código extra produzido pela equipe e que sejam necessários para replicar o experimento estão também disponíveis publicamente.

Repositório deste minicurso:
`https://github.com/LABORA-INF-UFG/SBRC2020-Minicurso3`.

## 3.7. Perspectivas futuras e Conclusão

Como descrevemos previamente, a academia já começou a investigar potenciais desafios para a próxima geração de redes móveis sem fio. Naturalmente, ainda há muita incerteza e várias previsões podem não se consolidar. Por outro lado, alguns temas começam a ser abordados, ainda que de maneira incipiente, no roteiro de lançamentos da 3GPP. Por exemplo, nos Lançamentos 16 e 17 estão previstas melhorias em URLLC e o uso de SON para tornar mais eficiência de redes 5G, sugerindo que previsões como MBRLLC e/ou mURLLC, assim como a ampla adoção de Aprendizado de Máquina (ML – *Machine Learning*) e Inteligência Artificial (AI – *Artificial Intelligence*), podem vir a se tornar realidade no futuro. A seguir, discutimos brevemente alguns tópicos abordados nos próximos dois lançamentos, previstos para junho de 2020 (*Release* 16) e setembro de 2021 (*Release* 17).

### Rede guiada por dados

A introdução da NWDAF no Lançamento 15 é um passo importante para adoção de ML e AI em rede 5G, porém é apenas o começo. A especificação completa do arcabouço de coleta e análise de dados deve ser concluída no Lançamento 16. A partir desse ponto, soluções baseadas em ML e AI poderão utilizar as informações coletadas pela NWDAF para realizar tarefas como previsão e otimização da mobilidade, detecção de anomalias, previsão de QoS e correlação de dados. Nesse contexto, alguns dos objetivos

para o Lançamento 17 são:

- Desempenho de rede previsível assistido pela NWDAF;
- Análise de dados orientada ao UE;
- Exposição da análise de dados da NWDAF para aplicações de usuário;
- NWDAF apoiando detecção de eventos anômalos e ajudando na análise de suas causas.

No longo prazo, o objetivo é usar técnicas de ML e AI para automatizar a gestão da rede com o mínimo de intervenção humana possível. É importante destacar que essa gestão deverá envolver diferentes tipos de redes (*i.e.*, redes heterogêneas) se conectando a um núcleo comum, baseado na arquitetura SBA.

**Melhorias para domínios verticais**

Conforme comentado anteriormente, redes 5G foram projetadas com atenção especial para domínios verticais (comumente chamados apenas de verticais), isto é, setores ou grupos de empresas específicas em que produtos ou serviços semelhantes são desenvolvidos, produzidos e fornecidos. Nesse contexto, no Lançamento 16 existem várias direções e contribuições, por exemplo:

- Suporte a comunicações sensíveis a tempo (TSC – *Time Sensitive Communication*);
- Redes não públicas (NPNs – *Non-Public Networks*), *i.e.*, redes privadas;
- Suporte para serviços do tipo rede local 5G (5G LAN);
- Serviços de localização avançados.

No Lançamento 17, estão previstas evoluções no suporte a NPNs e a serviços de proximidade (ProSe - *Proximity Sservices*), sendo esses também úteis em cenários de segurança pública.

**Evolução da segurança**

Segurança sempre foi uma preocupação em redes celulares, especialmente a partir de 3G com o uso mais amplo da infraestrutura para o transporte de dados e sua natural integração com a Internet. Portanto, a questão de segurança em redes 5G foi abordada desde seu projeto e tem sido um tema recorrente, sobretudo devido a importância estratégica que a 5G vem conquistando em termos econômicos e sociais. Existe uma lista de contribuições em segurança previstas para o Lançamento 16, da qual alguns itens relevantes são destacados a seguir:

- Suporte para NPNs com novos esquemas de autenticação;
- Opção de autenticação específica por fatia de rede (*network slice*), além da autenticação primária;

- Segurança avançada para Controle de Recursos de Rádio (RRC – *Radio Resource Control*) e sinalização NAS;
- Suporte para proteção de integridade no plano de usuário, abrangendo os três cenários de redes 5G, *i.e.*, eMBB, URLLC e mMTC.

**Outros avanços**

Há várias outras contribuições nos Lançamentos 16 e 17 que introduzem novas soluções, mas que também podem oferecer oportunidades para investigação em pesquisa. Isso ocorre porque determinadas soluções não estão completamente definidas, devido a fatores como complexidade do problema abordado ou impossibilidade de definir no padrão as especificidades que a solução terá que lidar. A seguir, listamos alguns temas desses dois Lançamentos (16 e 17) que, embora já tenham sido investigados, ainda apresentam questões em aberto, especialmente em um contexto prático e com potencial aplicação em larga escala.

- Avanços em V2X (*Vehicle-to-everything*) – formação de pelotão, direção autônoma e direção remota;
- Acesso a espectro não licenciado usando o 5G NR (*New Radio*);
- Compartilhamento Dinâmico do Espectro (DSS – *Dynamic Spectrum Sharing*);
- IoT em redes não terrestres (NTNs – *Non Terrestrial Networks*);
- Suporte a sistemas aéreos não tripulados (UAS – *Unmanned Aerial Systems*).

Finalmente, baseado no que foi apresentado nesse minicurso, podemos perceber a integração cada vez mais forte entre as áreas de Telecomunicação e Tecnologia da Informação. Por um lado, tem crescido a adoção de softwarização e modelo nativo em nuvem em sistemas 5G, *i.e.*, RAN e núcleo. Por outro lado, tem crescido o interesse em criar soluções em software que atendam as necessidades específicas de redes móveis sem fio, como computação de borda e sistemas para processamento de sinais. Essa integração ainda será largamente explorada nos próximos anos, para tornar o desenvolvimento ainda mais ágil e interoperável. Exemplo dessa mudança de visão é a distribuição (a partir do Lançamento 15) das especificações da 3GPP no formato de serialização de dados legíveis (YAML - *YAML Ain't Markup Language*) [65], permitindo a construção e desenvolvimento das interfaces padronizadas e com grande flexibilidade. Além disso, a utilização de tecnologias nativas de nuvem permitirá uma gerência e controle extremamente automatizados, com a mínima (ou até mesmo sem a) intervenção humana, que vem sendo chamado de ZSM (*Zero touch network & Service Management*) [66]. Essas evoluções devem alterar o modelo de negócio das operadoras de telecomunicação, permitindo ampliar relações B2C (*Business to Consumer*) e B2B (*Business to Business*).

## Referências

[1] Shafi, M. et al. 5G: A Tutorial Overview of Standards, Trials, Challenges, Deployment, and Practice. *IEEE Journal on Selected Areas in Communications*, v. 35, n. 6, p. 1201–1221, 2017.

[2] Kaloxylos, A. A Survey and an Analysis of Network Slicing in 5G Networks. *IEEE Communications Standards Magazine*, v. 2, n. 1, p. 60–65, 2018.

[3] RICHTER, F. *Global 5G Adoption to Take Off in 2021*. 2019. https://www.statista.com/chart/9604/5g-subscription-forecast/. [Último acesso: 31-mar-2020].

[4] World Economic Forum. *The Impact of 5G: Creating New Value across Industries and Society*. Jan 2020. https://www.weforum.org/whitepapers/the-impact-of-5g-creating-new-value-across-industries-and-society. [Último acesso: 31-mar-2020].

[5] 3GPP. *4G; Release 8*. 2009. https://www.3gpp.org/specifications/releases/72-release-8. [Último acesso: 26-fev-2020].

[6] 3GPP. *TR21.915: Technical Specification Group Services and System Aspects; Release 15 Description; Summary of Rel-15 Work Items (Release 15)*. [S.l.], 2018–12. Version 0.5.0.

[7] Abdelwahab, S. et al. Network function virtualization in 5G. *IEEE Communications Magazine*, v. 54, n. 4, p. 84–91, 2016.

[8] Chen, T. et al. Software defined mobile networks: concept, survey, and research directions. *IEEE Communications Magazine*, v. 53, n. 11, p. 126–133, 2015.

[9] FOUKAS, X. et al. Network slicing in 5G: Survey and challenges. *IEEE Communications Magazine*, v. 55, n. 5, p. 94–100, 2017.

[10] MADEMANN, F. The 5G System Architecture. *Journal of ICT Standardization*, v. 6, n. 3, p. 77–86, 2018.

[11] Palattella, M. R. et al. Internet of Things in the 5G Era: Enablers, Architecture, and Business Models. *IEEE Journal on Selected Areas in Communications*, v. 34, n. 3, p. 510–527, 2016.

[12] Lema, M. A. et al. Business Case and Technology Analysis for 5G Low Latency Applications. *IEEE Access*, v. 5, p. 5917–5935, 2017.

[13] Recommendation ITU-R M.2083-0. *IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond*. 2015. https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.2083-0-201509-I!!PDF-E.pdf.

[14] STALLINGS, W. *Data and computer communications*. 10. ed. [S.l.]: Prentice Hall, 2014.

[15] GAWAS, A. An overview on evolution of mobile wireless communication networks: 1G-6G. *International Journal on Recent and Innovation Trends in Computing and Communication*, v. 3, n. 5, p. 3130–3133, 2015.

[16] NGUYEN, V.-G. et al. SDN/NFV-based mobile packet core network architectures: A survey. *IEEE Communications Surveys & Tutorials*, IEEE, v. 19, n. 3, p. 1567–1602, 2017.

[17] VAEZI, M.; DING, Z.; POOR, H. V. *Multiple access techniques for 5G wireless networks and beyond*. [S.l.]: Springer, 2019.

[18] OJALA, P. et al. The adaptive multirate wideband speech codec: system characteristics, quality advances, and deployment strategies. *IEEE Communications Magazine*, IEEE, v. 44, n. 5, p. 59–65, 2006.

[19] GUILLOU, Y. L. et al. Highly integrated direct conversion receiver for GSM/GPRS/EDGE with on-chip 84-dB dynamic range continuous-time/spl Sigma//spl Delta/ADC. *IEEE journal of solid-state circuits*, IEEE, v. 40, n. 2, p. 403–411, 2005.

[20] BETTSTETTER, C.; VOGEL, H.-J.; EBERSPACHER, J. GSM phase 2+ general packet radio service GPRS: Architecture, protocols, and air interface. *IEEE communications Surveys*, IEEE, v. 2, n. 3, p. 2–14, 1999.

[21] CONDOLUCI, M.; MAHMOODI, T. Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges. *Computer Networks*, Elsevier, v. 146, p. 65–84, 2018.

[22] SUBRAMANYA, S.; YI, B. K. Mobile communications-an overview. *IEEE Potentials*, IEEE, v. 24, n. 5, p. 36–40, 2005.

[23] GHOSH, A. et al. 5G Evolution: A View on 5G Cellular Technology Beyond 3GPP Release 15. *IEEE Access*, IEEE, v. 7, p. 127639–127651, 2019.

[24] PUNZ, G. *Evolution of 3G networks: the concept, architecture and realization of mobile networks beyond UMTS*. [S.l.]: Springer Science & Business Media, 2009.

[25] 3GPP. *TP-03075: Overview of 3GPP Release 99*. 2003. `ftp://www.3gpp.org/TSG_T/TSG_T/TSGT_22/Docs/PDF/TP-030275.pdf`. [Último acesso: 22-apr-2020].

[26] NOHRBORG, M. *LTE Overview*. aug 2012. [Último acesso: 11-mar-2020]. Disponível em: <https://www.3gpp.org/technologies/keywords-acronyms/98-lte>.

[27] PARKVALL, S.; FURUSKAR, A.; DAHLMAN, E. Evolution of LTE toward IMT-advanced. *IEEE Communications Magazine*, IEEE, v. 49, n. 2, p. 84–91, 2011.

[28] SACHS, J. et al. 5G radio network design for ultra-reliable low-latency communication. *IEEE Network*, IEEE, v. 32, n. 2, p. 24–31, 2018.

[29] WANNSTROM, J. LTE-advanced. *Third Generation Partnership Project (3GPP)*, 2013.

[30] SCHMITT, P.; LANDAIS, B.; YANG, F. Y. *Control and User Plane Separation of EPC nodes (CUPS)*. aug 2012. [Último acesso: 25-mar-2020]. Disponível em: <https://www.3gpp.org/news-events/1882-cups>.

[31] Rangan, S.; Rappaport, T. S.; Erkip, E. Millimeter-Wave Cellular Wireless Networks: Potentials and Challenges. *Proceedings of the IEEE*, v. 102, n. 3, p. 366–385, 2014.

[32] Wang, X. et al. Millimeter Wave Communication: A Comprehensive Survey. *IEEE Communications Surveys Tutorials*, v. 20, n. 3, p. 1616–1653, 2018.

[33] Giordani, M. et al. A Tutorial on Beam Management for 3GPP NR at mmWave Frequencies. *IEEE Communications Surveys Tutorials*, v. 21, n. 1, p. 173–196, 2019.

[34] Zhang, D. et al. Fast Beam Tracking for Millimeter-Wave Systems Under High Mobility. In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. [S.l.: s.n.], 2019. p. 1–6.

[35] Samsung Electronics Co. *5G Core Vision – Samsung 5G Core Vol.1*. [S.l.], 2019.

[36] SAAD, W.; BENNIS, M.; CHEN, M. A Vision of 6G Wireless Systems: Applications, Trends, Technologies, and Open Research Problems. *CoRR*, abs/1902.10265, 2019. Disponível em: <http://arxiv.org/abs/1902.10265>.

[37] Giordani, M. et al. Toward 6G Networks: Use Cases and Technologies. *IEEE Communications Magazine*, v. 58, n. 3, p. 55–61, 2020.

[38] CHEN, Z. et al. A survey on terahertz communications. *China Communications*, IEEE, v. 16, n. 2, p. 1–35, 2019.

[39] Semiari, O. et al. Integrated Millimeter Wave and Sub-6 GHz Wireless Networks: A Roadmap for Joint Mobile Broadband and Ultra-Reliable Low-Latency Communications. *IEEE Wireless Communications*, v. 26, n. 2, p. 109–115, 2019.

[40] Peng, M. et al. Self-configuration and self-optimization in LTE-advanced heterogeneous networks. *IEEE Communications Magazine*, v. 51, n. 5, p. 36–45, 2013.

[41] Letaief, K. B. et al. The Roadmap to 6G: AI Empowered Wireless Networks. *IEEE Communications Magazine*, v. 57, n. 8, p. 84–90, 2019.

[42] Adadi, A.; Berrada, M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, v. 6, p. 52138–52160, 2018.

[43] Taleb Zadeh Kasgari, A.; Saad, W.; Debbah, M. Human-in-the-Loop Wireless Communications: Machine Learning and Brain-Aware Resource Management. *IEEE Transactions on Communications*, v. 67, n. 11, p. 7727–7743, 2019.

[44] Marotta, M. A. et al. Characterizing the Relation Between Processing Power and Distance Between BBU and RRH in a Cloud RAN. *IEEE Wireless Communications Letters*, v. 7, n. 3, p. 472–475, 2018.

[45] 3GPP. *TR38.801: 33rd Generation Partnership Project; Technical Specification Group Radio Access Network; Study on new radio access technology: Radio access architecture and interfaces (Release 14)*. [S.l.], 2017–03. Version 14.0.0.

[46] Gonzalez-Diaz, S. et al. Integrating fronthaul and backhaul networks: Transport challenges and feasibility results. *IEEE Transactions on Mobile Computing*, p. 1–18, 2019.

[47] FONSECA, F.; CORREA, S.; CARDOSO, K. Optimizing allocation and positioning in a disaggregated radio access network aware of paths through the core infrastructure. In: *Anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.: s.n.], 2019. p. 791–804.

[48] Karnouskos, S. et al. A SOA-based architecture for empowering future collaborative cloud-based industrial automation. In: *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*. [S.l.: s.n.], 2012. p. 5766–5772. ISSN 1553-572X.

[49] Imadali, S.; Bousselmi, A. Cloud Native 5G Virtual Network Functions: Design Principles and Use Cases. In: *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*. [S.l.: s.n.], 2018. p. 91–96.

[50] HEDMAN, P. et al. *5G Core Networks: Powering Digitalisation*. [S.l.]: Elsevier Science & Technology, 2019. ISBN 9780081030097.

[51] 3GPP. *TS133.185: 3rd Generation Partnership Project Technical Specification Group Services and Systems Aspects Release 14 Description*. [S.l.], 2017–10. Version 14.1.0.

[52] TOSKALA, A.; POIKSELKÄ, M. 5G Architecture. *5G Technology: 3GPP New Radio*, Wiley Online Library, p. 67–86, 2020.

[53] ETSI. *TS 123 501: System architecture for the 5G System (5GS)(3GPP TS 23.501 version 15.5.0 Release 15*. [S.l.], 2019. Version 15.5.0.

[54] Choyi, V. K. et al. Network slice selection, assignment and routing within 5G Networks. In: *IEEE Conference on Standards for Communications and Networking (CSCN)*. [S.l.: s.n.], 2016. p. 1–7.

[55] PARKVALL, S. et al. NR: The new 5G radio access technology. *IEEE Communications Standards Magazine*, IEEE, v. 1, n. 4, p. 24–30, 2017.

[56] 3GPP. *5G; Access to the 3GPP 5G Core Network (5GCN) via non-3GPP access networks (Release 15)*. 2019. https://www.etsi.org/deliver/etsi_ts/124500_124599/124502/15.02.00_60/ts_124502v150200p.pdf. [Último acesso: 03-fev-2020].

[57] WBA and NGMN Alliance. *RAN Convergence Paper*. 2019. https://wballiance.com/wp-content/uploads/2019/01/RAN-Convergence-Paper-2019.pdf. [Último acesso: 03-fev-2020].

[58] Navarro-Ortiz, J. et al. Integration of LoRaWAN and 4G/5G for the Industrial Internet of Things. *IEEE Communications Magazine*, v. 56, n. 2, p. 60–67, Feb 2018.

[59] Yasmin, R. et al. On the integration of LoRaWAN with the 5G test network. In: *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. [S.l.: s.n.], 2017. p. 1–6.

[60] OLSSON, M. et al. *EPC and 4G packet networks: driving the mobile broadband revolution*. [S.l.]: Academic Press, 2013.

[61] SEMTECH. *Semtech Corporation*. 2020. `https://www.semtech.com/`. [Último acesso: 22-abril-2020].

[62] Shanmuga Sundaram, J. P.; Du, W.; Zhao, Z. A survey on lora networking: Research problems, current solutions, and open issues. *IEEE Communications Surveys Tutorials*, v. 22, n. 1, p. 371–388, 2020.

[63] JungWoon Lee et al. Risk analysis and countermeasure for bit-flipping attack in lorawan. In: *International Conference on Information Networking (ICOIN)*. [S.l.: s.n.], 2017. p. 549–551.

[64] HAXHIBEQIRI, J. et al. A Survey of LoRaWAN for IoT: From Technology to Application. *Sensors*, v. 18, p. 3995, 11 2018.

[65] 3GPP. *OpenAPI*. feb 2020. [Último acesso: 26-abril-2020]. Disponível em: <https://www.3gpp.org/FTP/Specs/archive/OpenAPI/>.

[66] ETSI. *Zero-touch network and Service Management (ZSM); Terminology for concepts in ZSM*. [S.l.], 2019. V1.1.1. Disponível em: <https://www.etsi.org/technologies/zero-touch-network-service-management>.

# Capítulo 4

# Aprendizado Profundo em Redes Desafiadoras: Conceitos e Aplicações

Kaylani Bochie (UFRJ), Mateus S. Gilbert (UFRJ),
Luana Gantert (UFRJ), Mariana S. M. Barbosa (UFRJ),
Dianne S. V. Medeiros (UFF) e Miguel Elias M. Campista (UFRJ)

***Resumo***

*Este minicurso tem por objetivo apresentar de forma clara e concisa os conceitos técnicos, desafios e aplicações de técnicas de aprendizado profundo nas chamadas Redes Desafiadoras. Essas redes são caracterizadas pelo excesso de dados produzidos e consequente complexidade de operação. Diferente dos minicursos anteriores, este minicurso foca na revisão da literatura relacionada à aplicação de técnicas de aprendizado profundo na solução de problemas complexos das Redes Desafiadoras. Para tanto, este minicurso apresenta e discute os principais conceitos e algoritmos relacionados ao aprendizado de máquina, com foco no aprendizado profundo. Em seguida, a literatura relativa à aplicação desses algoritmos em redes, como redes de Internet das Coisas e de sensores, redes sem fio móveis, redes industriais e redes veiculares é revisada. Este minicurso também apresenta viés experimental, oferecendo um estudo de caso como exercício prático para que os participantes tenham a oportunidade de se debruçar sobre um problema real.*

## 4.1. Introdução

A complexidade crescente dos sistemas computacionais é inegável, dado o aumento vertiginoso na quantidade de dados a serem analisados para a oferta eficiente de serviços. Em redes de computadores, esse fenômeno não poderia ser diferente, sendo consequência direta da escalada no número de usuários, cenários de aplicação, dinâmica dos nós da rede, fontes de dados, etc. Considerar todos esses parâmetros dificulta a análise e manutenção da Qualidade de Serviço (*Quality of Service* – QoS) e o emprego de protocolos e algoritmos clássicos que não possuem a capacidade de aprendizado. Na Internet das Coisas (*Internet of Things* – IoT), por exemplo, a massa de dados proveniente de sensores diversos é um obstáculo adicional. Nas redes sem fio, a dinamicidade da rede

dificulta a distinção entre comportamento esperado e anomalias. Já em redes industriais, um fator crítico é a disponibilidade que deve levar em conta diferentes motivos para falhas e mitigar seus efeitos. Por fim, nas redes veiculares, a previsão da ocorrência e duração de contatos para transferência eficiente de dados é algo que ainda requer esforços em pesquisa, principalmente na ausência de infraestrutura fixa de comunicação. Neste minicurso, as redes caracterizadas pela geração de um grande volume de dados e pela inerente complexidade de operação são chamadas de "*Redes Desafiadoras*".

O conceito de Aprendizado de Máquina (*Machine Learning*) e suas técnicas correlatas existem há décadas [Osherson et al., 1991]. O ressurgimento do interesse sobre essas técnicas foi impulsionado pela contínua evolução do *hardware* e do *software* que permitiu o uso de ferramentas que exigem processamento intensivo. No contexto atual, o aprendizado de máquina aparece como uma ferramenta alternativa aos algoritmos tradicionais, baseados em regras, que possibilita a análise de diferentes cenários em Redes Desafiadoras sem que as regras de análise sejam explicitamente programadas. O aprendizado de máquina consiste em obter uma representação matemática que modele o comportamento de uma função através de um processo chamado de treinamento. No treinamento, a partir de amostras (*samples*), normalmente com múltiplos atributos (*features*), o modelo tem seus parâmetros ajustados de forma a conseguir prever um conjunto inédito de amostras, exercer uma ação de modo autônomo ou extrair informações relevantes. Durante o aprendizado, é possível que existam alvos (*targets*), que as saídas do modelo devem prever. O bom desempenho do modelo depende de forma crucial da escolha de atributos. Assim, a seleção de atributos acaba se tornando uma extensa área de pesquisa no campo de aprendizado de máquina [Mao et al., 2018]. Outro ponto importante é a capacidade do modelo matemático de "compreender" um problema complexo, capacidade que deve vir acompanhada de rápida resposta e baixa sensibilidade a variações.

Tendo em vista a importância da seleção de atributos e a crescente complexidade nos problemas encontrados em Redes Desafiadoras, o Aprendizado Profundo (*Deep Learning*) tem sido revisitado. Pesquisas nessa área datam dos anos 40, passando por três grandes picos de relevância referenciados por outros nomes: *cybernetics* entre as décadas de 40 e 60, *connectionism* entre os anos 80 e 90 e, por fim, aprendizado profundo a partir de meados dos anos 2000 [Goodfellow et al., 2016]. Nos primórdios, a capacidade computacional era um problema crítico, tornando-se um entrave até a segunda fase. Com o rápido desenvolvimento tecnológico na fase atual, o poder computacional deixa de ser um problema tão relevante para a popularização do aprendizado profundo. Como consequência, há um crescente uso das técnicas e dos algoritmos, mesmo aqueles desenvolvidos nas duas fases precedentes, que permeiam as aplicações atuais em diversas áreas de pesquisa. A popularidade desse tipo de aprendizado se deve ao uso de modelos matemáticos mais complexos que melhoram a seleção de atributos e, consequentemente, aprimoram os resultados alcançáveis. Além disso, através da introdução de diversas camadas de processamento, o aprendizado profundo adapta os modelos empregados à dificuldade do problema. Tipicamente, o aprendizado profundo requer modelos que capturem não-linearidades do problema, tornando a modelagem relativamente mais complexa.

Este minicurso tem por objetivo apresentar de forma clara e concisa os conceitos técnicos, desafios e aplicações de técnicas de aprendizado profundo nas chamadas Redes Desafiadoras. Diferentemente dos minicursos apresentados em simpósios brasileiros

anteriores [Comarela et al., 2019, Medeiros et al., 2019], o foco deste minicurso está na revisão não exaustiva da literatura relacionada à aplicação de técnicas de aprendizado profundo em um determinado conjunto de redes com características marcantes. Para isso, primeiramente, o minicurso apresenta e discute os principais conceitos e algoritmos relacionados ao aprendizado de máquina com foco no aprendizado profundo. Em seguida, quatro tipos de **Redes Neurais Artificiais (*Artificial Neural Networks* – ANNs)** são apresentados. O foco nas redes neurais é dado, visto que muitos dos trabalhos em aprendizado profundo aplicam esse tipo de abordagem. Isso é observado através da aplicação desses algoritmos em redes IoT e de sensores, redes sem fio móveis, redes industriais e redes veiculares que, em comum, possuem desafios relacionados majoritariamente à dinamicidade e às limitações de armazenamento e processamento de dados em seus nós. A ideia é capturar, a partir desses trabalhos, abordagens que sejam comuns a um dado tipo de rede ou mesmo a todas como forma de nortear a pesquisa na área. Ao final, uma atividade prática é oferecida aos leitores para fixação do aprendizado adquirido.

Este minicurso está organizado como segue. A Seção 4.2 apresenta os principais conceitos relacionados ao aprendizado de máquina, e principais algoritmos e modelos de aprendizado profundo. A Seção 4.3 aborda aplicações baseadas em aprendizado profundo para solucionar problemas em Redes Desafiadoras. Ainda, essa seção apresenta trabalhos de detecção de intrusão e anomalias, um problema transversal a todas as redes desafiadoras. A Seção 4.4 apresenta ferramentas frequentemente utilizadas na análise de dados de redes de computadores e descreve uma atividade prática focada na detecção de ataques com base em redes neurais profundas. Por fim, a Seção 4.5 conclui este minicurso e apresenta as perspectivas e problemas em aberto relacionados ao tema do minicurso.

## 4.2. Conceitos de Aprendizado Profundo

Esta seção familiariza o leitor com os termos e técnicas básicas relevantes para a continuidade do minicurso. A seção apresenta conceitos fundamentais de aprendizado de máquina necessários para a compreensão do aprendizado profundo, as principais diferenças entre aprendizado de máquina e profundo e, por fim, os diferentes tipos de redes neurais profundas, que são as principais estruturas utilizadas no aprendizado profundo.

### 4.2.1. Conceitos Básicos de Aprendizado de Máquina

O aprendizado de máquina surge como um paradigma no qual os algoritmos são capazes de extrair informações dos dados disponibilizados sem serem explicitamente programados [Medeiros et al., 2019]. Dessa forma, é possível reduzir a necessidade de conhecimento prévio sobre o domínio no qual são empregados [Deisenroth et al., 2019]. Essa característica torna as técnicas de aprendizado de máquina atraentes para problemas que envolvem grandes massas de dados, possivelmente de difícil formalização, cuja programação manual de algoritmos tradicionais pode ser custosa ou até mesmo impossível [Goodfellow et al., 2016]. O aumento da popularidade do uso das técnicas de aprendizado de máquina justifica-se pelo casamento entre os cenários atuais, que fazem uso de massas de dados cada vez maiores, e pela evolução na infraestrutura computacional, que oferece o aporte necessário para a execução de técnicas mais avançadas.

O pré-requisito fundamental para os algoritmos que compõem o paradigma do

aprendizado de máquina é a capacidade de *"aprender"*. Mitchell define que um programa está aprendendo se seu desempenho na execução de um conjunto de tarefas, medido a partir de uma métrica de interesse, melhora com a experiência desse conjunto [Mitchell, 1997]. Os algoritmos tiram vantagem de possíveis padrões e estruturas presentes nos dados do problema para ajustar seus parâmetros e, como consequência, melhorar o seu desempenho na tarefa apresentada. Dessa forma, o caráter autônomo dessa abordagem fica evidente, visto que os parâmetros do algoritmo são definidos pela interação do próprio algoritmo com o conjunto de dados. O processo no qual o algoritmo tem seus parâmetros ajustados a fim de aprender a realizar uma tarefa é chamado de *treinamento*, que pode ser feito de diferentes maneiras, dependendo da construção dos dados em relação ao mapeamento entre entradas e saídas. Como consequência, os algoritmos de aprendizado de máquina podem ser divididos em *supervisionado*, *não supervisionado* e *por reforço*.

- **Aprendizado supervisionado**: os algoritmos têm acesso a um conjunto de dados rotulados, ou seja, existem exemplos do mapeamento entre entradas e saídas. A presença dos rótulos possibilita que os algoritmos ajustem seus parâmetros para reproduzirem as mesmas saídas caso entradas semelhantes sejam apresentadas. Em uma analogia ao aprendizado humano, o algoritmo de aprendizado supervisionado tem acesso às respostas corretas das perguntas de um teste e aprende com o acesso a essas respostas. As respostas corretas das perguntas do teste são análogas ao mapeamento entre entradas e saídas promovido pelo rotulamento dos dados.

- **Aprendizado não supervisionado**: o conjunto de dados carece de rótulos, não existindo um mapeamento entre entradas e saídas. Nesse cenário, os algoritmos buscam relações e características presentes no conjunto de dados que possam ser exploradas para classificar internamente os elementos. Essa classificação pode levar a grupos de dados que compartilhem características semelhantes ou a grupos de dados que possuam algum tipo de correlação. As relações inferidas são mensuradas por métricas que verificam se a classificação obtida é adequada, possibilitando que os algoritmos ajustem os seus parâmetros. Analogamente ao aprendizado humano, os algoritmos de aprendizado não supervisionado avaliam padrões, assim como um bebê observa o comportamento e as características que definem uma pessoa conhecida, por exemplo. Observando os padrões de comportamento e características de uma pessoa qualquer, o bebê é capaz de associar aquele conjunto de entradas de dados à saída que determina se a pessoa é conhecida ou não. Não é necessário, nesse caso, que seja informado previamente ao bebê que ele conhece a pessoa.

- **Aprendizado por reforço**: os algoritmos se baseiam em um modelo de recompensas e punições à medida que o modelo interage com o ambiente onde está inserido. Assim, em vez de existir um mapeamento direto entre entradas e saídas, os resultados são obtidos a partir da realimentação (*feedback loop*) entre o sistema de aprendizado e o ambiente. A cada iteração, as ações disponíveis são apresentadas ao modelo no seu estado atual e, após a mudança de estado, recebe um sinal de reforço. De forma similar ao aprendizado humano, os algoritmos de aprendizado por reforço buscam instigar um conjunto de comportamentos desejados, como o uso correto de talheres para uma criança, através de recompensas, como palavras de re-

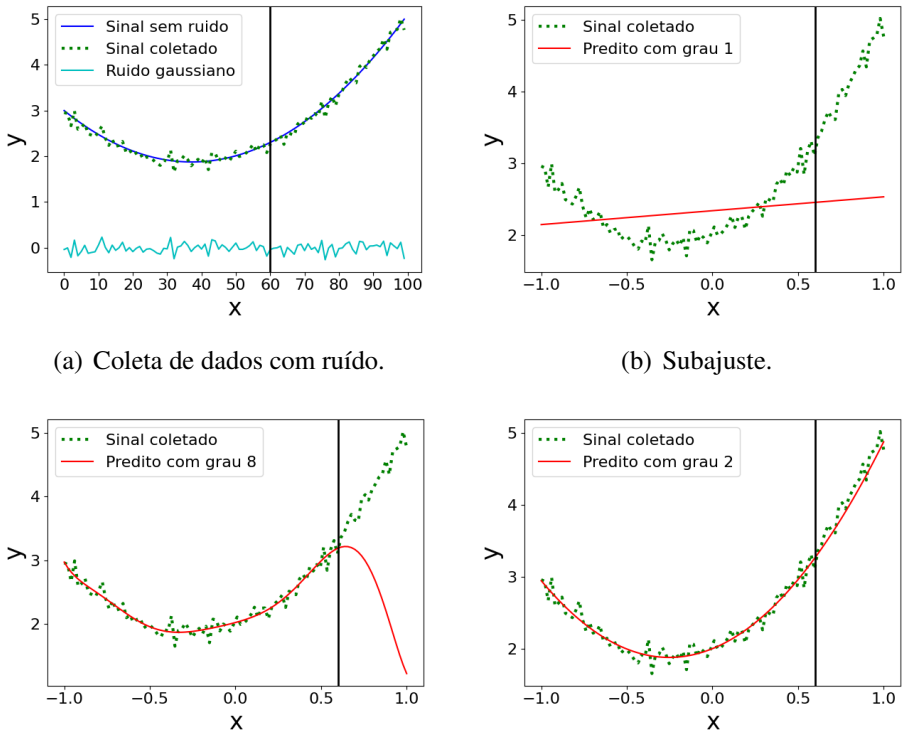
força ou um prêmio. O objetivo dessa abordagem é escolher ações que maximizem a recompensa a longo prazo [Kaelbling et al., 1996].

Ressalta-se que a divisão apresentada é uma forma aproximada de compreender a função dos algoritmos e demonstrar resumidamente o emprego de cada um deles. Porém, outros paradigmas de aprendizado são possíveis. Em especial, alguns desses paradigmas surgem da combinação das classes descritas anteriormente. O **aprendizado semissupervisionado** é um desses casos, que combina elementos dos aprendizados supervisionado e não supervisionado. Esse paradigma é empregado, por exemplo, para rotular conjuntos de dados não rotulados, ou com rótulos faltantes. Para tal, o conjunto de dados é usado como entrada da parte não supervisionada do algoritmo para que este gere os rótulos. Uma vez que todos os dados estão rotulados, a parte supervisionada é usada para classificação.

Além dos tipos de algoritmos, existem conceitos fundamentais que dizem respeito à efetividade de um programa após a modelagem e o próprio aprendizado. Um pré-requisito essencial para o aprendizado de máquina é a capacidade de *generalização*, isto é, o poder de responder de maneira adequada a situações que o algoritmo não teve acesso em seu aprendizado. Para verificar essa capacidade, as amostras dos dados coletados são comumente divididas em três conjuntos: *conjunto de treinamento, conjunto de validação e conjunto de teste*. O **conjunto de treinamento** contém as amostras que o algoritmo efetivamente utiliza para ajustar os seus parâmetros. Portanto, é durante o treinamento que os modelos de aprendizado acessam parte dos dados para ajustar seus parâmetros internos. O processo de treinamento propriamente dito pode diferir de acordo com os modelos de aprendizado adotados. No treinamento, o modelo de aprendizado passa por diversas etapas de atualização de parâmetros e verificação de desempenho. O **conjunto de validação** é usado para acompanhar o progresso no aprendizado, servindo para medir o erro, ou o custo, associado à configuração atual do algoritmo. O modelo utiliza o desempenho atual sobre os dados para alterar seus parâmetros, por exemplo a acurácia, para modelos de classificação, ou o Erro Médio Quadrático (*Mean Squared Error* – MSE) para modelos de regressão. A partir dessa medição, verifica-se se há margem para melhorar o algoritmo ou se o aprendizado deve ser encerrado. O **conjunto de teste** contém amostras inéditas que não foram utilizadas no treinamento e é usado para avaliar o comportamento do modelo já ajustado. Portanto, a avaliação do desempenho do algoritmo é feita a partir do conjunto de teste. É muito importante que o usuário do modelo apenas utilize o conjunto de teste após o ajuste completo do algoritmo feito nos conjuntos de treino e validação. Visto que reajustar características do modelo baseado nos resultados do conjunto de teste provoca **vazamento de dados** (*data leakage*) e provoca a criação de um modelo com desempenho normalmente superior, porque os dados usados durante o treino são usados também para avaliar o desempenho, o que invalida a generalização.

O desempenho de um algoritmo treinado, determinado quanto a sua capacidade de generalização, pode ser caracterizado por dois fatores fundamentais: subajuste (*underfitting*) e sobreajuste (*overfitting*). Durante o treinamento, tanto os casos de subajuste quanto os de sobreajuste devem ser evitados. Os conceitos de subajuste e sobreajuste estão ilustrados na Figura 4.1. A Figura 4.1(a) mostra um conjunto de dados fictício usado para treinar um algoritmo de aprendizado para regressão. Uma coleta de dados naturalmente é contaminada por ruído. Assim, os dados estudados representam um polinômio de

grau 2 com adição de ruído gaussiano, conforme ilustrado pelos pontos verdes. Durante o treinamento, o algoritmo tem acesso aos pontos coletados à esquerda da linha vertical preta e os pontos à direita dessa linha são as amostras inéditas usadas durante o teste.




(a) Coleta de dados com ruído.

(b) Subajuste.

(c) Sobreajuste.

(d) Complexidade apropriada.

**Figura 4.1. Um algoritmo de regressão utilizado para aproximar (a) uma curva polinomial de grau 2 com ruído aditivo, pode apresentar subajuste ou sobreajuste. Os dados à esquerda da linha vertical são usados para treinamento, e o restante para teste. (b) No subajuste, o modelo não é parametrizado de forma adequada e prediz dados de teste com baixa acurácia. (c) No sobreajuste, o modelo é bem ajustado aos dados de treinamento, mas não é capaz de predizer dados inéditos. (d) Um modelo de complexidade apropriada apresenta baixo erro no treinamento e prediz dados inéditos com elevada acurácia.**

O **subajuste** ocorre quando o algoritmo não está ajustado para executar a tarefa adequadamente, apresentando um elevado erro no treinamento. Nessa situação, o algoritmo falha na execução da tarefa designada pois o modelo definido por seus parâmetros não está ajustado apropriadamente. Um exemplo de subajuste é um algoritmo ajustado para aproximar pontos de um polinômio de ordem 2 por uma reta, como pode ser visto na Figura 4.1(b). Nessa situação é desejável continuar o treinamento, pois a experiência obtida das amostras de treino é benéfica para a melhora do algoritmo. Como é de se esperar, o erro ao longo do treinamento tende assintoticamente a um valor mínimo. Quanto maior a exposição do algoritmo às amostras de treinamento, melhor é o resultado obtido ao tratá-las. Porém, existe um momento no qual o treinamento excessivo do modelo culmina em um pior desempenho para amostras inéditas. Esse efeito é conhecido como **sobreajuste**. Quando isso ocorre, pode-se dizer que o algoritmo se especializa de tal forma que aprende as particularidades do conjunto de amostras utilizado durante o treinamento. Isso

faz com que o modelo apresente um desempenho ruim quando aplicado em amostras que diferem, mesmo que minimamente, do conjunto usado no treinamento. Na Figura 4.1(c) é possível observar que o modelo é ajustado de tal forma que as múltiplas amostras de um polinômio de grau 2, que possui flutuações naturais durante a coleta, são descritas através de um polinômio de grau 8 que apresenta um melhor ajuste às amostras de treino. Nota-se que tanto no sobreajuste (Figura 4.1(c)) quanto no subajuste (Figura 4.1(b)) o algoritmo falha em prever o valor das funções ao ser apresentado a novas amostras. Visto que ambos os casos são prejudiciais ao desempenho do algoritmo, o objetivo do treinamento é encontrar o ponto ótimo que maximiza o desempenho do modelo em amostras inéditas. A Figura 4.1(d) apresenta um modelo com complexidade apropriada para a predição de novas amostras no caso do regressor polinomial.

As técnicas de *regularização* podem ser usadas para que um algoritmo tenha maior margem para minimizar o erro de aprendizado sem que ocorra uma situação de sobreajuste. A função dos regularizadores é aumentar a capacidade de generalização do algoritmo, ou seja, fazer com que o desempenho do modelo no conjunto de teste aumente com o tempo de treinamento. Um exemplo de regularizador, no contexto de um regressor polinomial, é o acréscimo de um termo à função custo que controla a magnitude dos coeficientes do polinômio, impedindo que eles assumam valores muito altos.

Como a saída do modelo de aprendizado é uma função de seus parâmetros internos e dos valores inseridos, e as entradas não podem ser alteradas pelo algoritmo, o ajuste dos parâmetros deve ser feito para criar a função de transferência correta. Dessa forma, torna-se importante distinguir *parâmetros* de *hiperparâmetros*, que fazem parte do processo de aprendizado. Os **parâmetros** são valores aos quais o algoritmo tem acesso e é capaz de alterar, isto é, são valores que podem ser ajustados pelo algoritmo durante o treinamento. Já os **hiperparâmetros** são externos ao programa, sendo definidos *a priori* com a função de auxiliar o aprendizado, como o grau de um polinômio em um regressor polinomial. Os hiperparâmetros podem ser determinados pelo desenvolvedor ou com o auxílio de outro algoritmo de aprendizado de máquina, contanto que não possam ser ajustados pelo programa original. Em suma, as variáveis que o algoritmo pode alterar são chamadas de parâmetros, enquanto aquelas que ele não tem acesso são os hiperparâmetros. A escolha e o ajuste de hiperparâmetros devem ser feitos com o objetivo de maximizar o desempenho do modelo em uma tarefa real. É comum que os hiperparâmetros, quando ajustados de maneira manual, sejam determinados através de tentativa e erro. Porém, existem heurísticas para a escolha dos hiperparâmetros em função do problema abordado e do modelo escolhido [Wu et al., 2019, Neary, 2018].

### 4.2.2. Aprendizado Profundo: Um Caso de Aprendizado de Máquina

No aprendizado de máquina tradicional, alguns problemas requerem pré-tratamento dos dados que, por sua vez, exigem conhecimento prévio do desenvolvedor [LeCun et al., 2015]. Um exemplo é o desenvolvimento de uma aplicação que mapeia dados não lineares em uma representação linear, necessária para alguns algoritmos de aprendizado de máquina. Uma forma de contornar essa necessidade é fazer com que o algoritmo aprenda não só a mapear os dados de entrada tratados para a saída, como também a representação necessária para viabilizar tal mapeamento. Além disso, outro problema comum é encontrado quando o aprendizado de tarefas complexas ou conceitos abstratos são necessários,

onde o aprendizado da representação é quase tão difícil quanto o aprendizado da função em si [Goodfellow et al., 2016]. Nessa direção, o aprendizado profundo é uma área do aprendizado de máquina caracterizada pela extração dessas representações complexas a partir de representações mais simples. As soluções mais simples são organizadas em uma hierarquia cujos diferentes níveis se complementam para a composição de informações complexas [Goodfellow et al., 2016].

A divisão de um problema complexo em problemas menores permite que porções do algoritmo se especializem em tarefas mais simples, que depois são recombinadas em etapas posteriores para viabilizar a resolução da tarefa mais complexa. No reconhecimento de imagens, por exemplo, um algoritmo pode ser subdividido em módulos responsáveis pela identificação de determinadas partes de uma imagem, como as bordas. Posteriormente os módulos de identificação de borda podem ser combinados para identificar molduras e assim sucessivamente, até que a combinação dos módulos esteja especializada o suficiente para identificar uma figura. Assim, o modelo de aprendizado é dividido em múltiplas camadas de processamento. As camadas são organizações paralelas das estruturas de processamento básicas, que permitem a extração de informações progressivamente mais complexas à medida em que os dados são processados através das múltiplas camadas intermediárias, de forma hierárquica. Apesar de a definição de aprendizado profundo englobar quaisquer modelos de aprendizado que seguem essa abordagem hierárquica de processamento, o termo é comumente utilizado para fazer referência às **Redes Neurais Profundas** (*Deep Neural Networks* – DNNs). Existem outras abordagens aplicadas às Redes Desafiadoras, como o Aprendizado Profundo por Reforço (*Deep Reinforcement Learning* – DRL), que une redes neurais profundas a arquiteturas de aprendizado por reforço. No entanto, este minicurso foca nos diversos tipos existentes de DNN.

### 4.2.3. Redes Neurais Profundas

O aprendizado profundo engloba estruturas diversas que oferecem múltiplas camadas de processamento, porém as **redes neurais** concentram a maior parte dos esforços de pesquisa e dos algoritmos utilizados. Surgindo originalmente como uma tentativa de simular estruturas neuronais biológicas, a rede neural é uma estrutura composta por células computacionais simples, chamadas **neurônios**, massivamente interconectadas em uma estrutura paralelamente distribuída capaz de armazenar e processar informações para exercer uma tarefa [Haykin, 1994].

A estrutura típica de um neurônio pode ser observada na Figura 4.2. Tratando a entrada como um vetor $\boldsymbol{x}$, suas componentes $x_1, x_2 ... x_m$ correspondem às entradas do neurônio que são combinadas em uma soma ponderada, na qual esses valores são escalados utilizando seus respectivos pesos, representados por $w_{k1}$, $w_{k2}$ ... $w_{kn}$. O limiar de ativação inerente (*bias*) é introduzido como um grau de liberdade adicional para manipular a saída do neurônio, permitindo ao neurônio um melhor ajuste. O resultado da soma entre o limiar de ativação e as entradas multiplicadas pelos **pesos** produz o potencial de ativação $\upsilon_k$. A saída, normalmente indicada pelo vetor $y_k$, corresponde ao potencial de ativação transformado pela função de ativação $\varphi(.)$. A função $\varphi(.)$ é normalmente uma função não-linear do tipo $f(x) = max(x, 0)$ ou uma função do tipo **softmax**, que mapeia a saída em um intervalo entre 0 e 1.

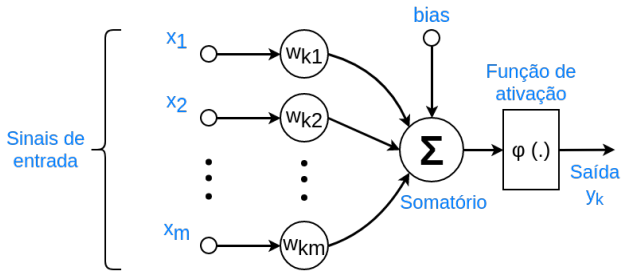


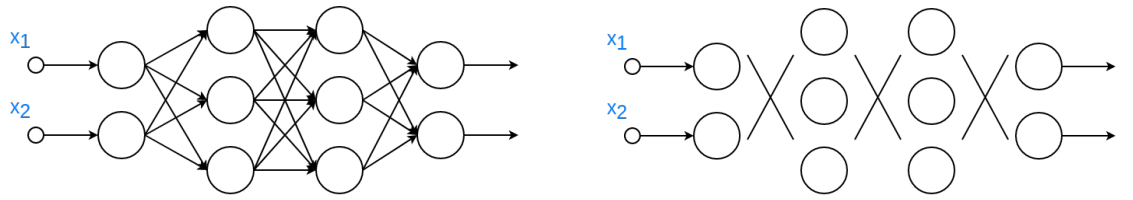

**Figura 4.2. Estrutura básica de um neurônio. O vetor de entrada é multiplicado por pesos. Os elementos ponderados e um limiar de ativação (*bias*) são somados e o potencial de ativação resultante alimenta uma função de ativação. A saída é o resultado da transformação do potencial de ativação.**

Nas redes neurais, o aprendizado ocorre com o ajuste dos pesos (valores das conexões) entre os neurônios e dos limiares de ativação (*biases*). Esses parâmetros são alterados durante a fase de treinamento. Os pesos são utilizados para calcular a taxa de crescimento da função que o algoritmo tenta modelar, enquanto os limiares de ativação são necessários para deslocar a saída da função. Uma função linear do tipo $y = \boldsymbol{\omega}^T \times \boldsymbol{x} + b$ pode ser modelada escolhendo apropriadamente os pesos ($\boldsymbol{\omega}$) e o limiar de ativação ($b$), por exemplo. Para cada amostra utilizada no treinamento, o modelo calcula o valor da saída para os valores atuais dos pesos e limiares de ativação, e compara o resultado com o valor esperado (alvo) da amostra. Uma função custo (*cost function*), calculada a partir de uma função de perda (*loss function*) aplicada a diversas amostras, é utilizada para quantificar o erro encontrado para a configuração atual do modelo e um **vetor gradiente**. Isto é, um vetor que contém a variação da saída do modelo em relação a cada um de seus parâmetros é computado para que os pesos sejam atualizados. O modelo atualiza os pesos e limiares de ativação no sentido contrário do vetor gradiente, buscando minimizar a função custo de acordo com uma taxa fixa ou passo, a taxa de aprendizado (*learning rate*).

Os neurônios nas redes neurais são normalmente organizados em grupos de unidades de processamento chamados de **camadas**, que por sua vez são organizadas em uma cadeia. A primeira e a última camada são chamadas de camada de entrada e saída, enquanto as demais são as camadas ocultas (ou internas). As redes neurais profundas se caracterizam pela composição de múltiplas camadas ocultas, como ilustrado na Figura 4.3(a). A Figura 4.3(b) apresenta uma representação simplificada e equivalente à Figura 4.3(a), normalmente utilizada para fins de ilustração. No aprendizado profundo, o aumento no número de camadas ocultas permite que cada uma delas se especialize em identificar um conjunto de características em particular. As camadas mais próximas da camada de entrada são responsáveis por identificar as características mais primitivas, como arestas em uma imagem, enquanto as seguintes combinam essas informações para identificar padrões mais complexos, como animais ou semáforos na mesma imagem. São diversos os tipos de redes neurais existentes. Neste minicurso, são discutidas as **Redes Neurais sem Realimentação**, **Convolucionais**, **Recorrentes** e **Autoassociativas**.

### Redes Neurais sem Realimentação

Nas Redes Neurais sem Realimentação (*Feedforward Neural Networks*), a informação é passada entre camadas em apenas um sentido a partir da entrada. O objetivo




(a) Representação tradicional de uma rede neural com duas camadas ocultas.

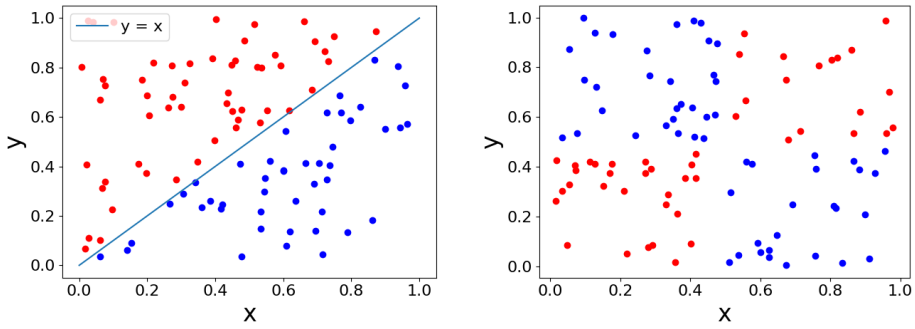
(b) Representação simplificada equivalente de uma rede neural com duas camadas ocultas.

**Figura 4.3. Um exemplo de rede neural com duas camadas ocultas, representada de forma (a) tradicional e (b) simplificada. Na simplificação, a ligação entre camadas adjacentes é substituída por um "X".**

desse tipo de rede é aproximar uma função qualquer, definindo um mapeamento do tipo $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$, onde a entrada $\boldsymbol{x}$ é conhecida e os parâmetros $\boldsymbol{\theta}$ são ajustados para oferecer a melhor aproximação da função desejada [Goodfellow et al., 2016]. A rede age, então, como uma composição de funções, na qual cada camada representa uma função da composição. Um dos exemplos mais conhecidos dessas redes é o **Perceptron de Múltiplas Camadas** (*Multilayer Perceptron* – MLP), que usualmente é construído como uma rede totalmente conectada (*fully connected network*) na qual todos os neurônios de uma camada são conectados aos neurônios da camada seguinte.

Uma camada da rede neural sem realimentação pode ser descrita através de uma função de transferência do tipo $\boldsymbol{h}_i = \varphi_i(\boldsymbol{W}_i^T \boldsymbol{h}_{i-1} + \boldsymbol{b}_i)$, onde $\boldsymbol{h}_i$, $\boldsymbol{h}_{i-1}$, $\boldsymbol{b}_i$ e $\varphi_i$ são, respectivamente, a saída da camada atual, a saída da camada anterior ou entrada da camada atual, o limiar de ativação da camada atual e a função de ativação de uma dada camada. O termo $\boldsymbol{W}_i^T$ é a matriz transposta que contém os pesos entre a camada anterior e a atual. Como esperado, nos casos das camadas de entrada e saída, a entrada da primeira é $\boldsymbol{h}_0 = \boldsymbol{x}$ e a saída da última, $\boldsymbol{h}_n = \boldsymbol{y}$. Nas camadas ocultas, a função de ativação $\varphi_i$ é usualmente não-linear, porém na última camada, a função pode não conter essa propriedade. O uso de funções não-lineares nas camadas internas é o que confere à rede uma maior capacidade de representação. A necessidade da função $\varphi_i$ ser não-linear é consequência da incapacidade da rede sem realimentação como um todo ser não-linear, caso todas as funções $\varphi_i$ fossem lineares. Nesse caso, a rede neural não seria capaz de resolver um problema que não é linearmente separável. Um exemplo é a incapacidade de modelos lineares em representar a função *XOR*. É possível observar na Figura 4.4(a) que um modelo linear é capaz de separar todos os valores da saída da função utilizando apenas uma reta, em contraste com o problema representado na Figura 4.4(b), que não é linearmente separável.

Uma rede neural sem realimentação é um aproximador universal. Isso quer dizer que essas redes têm a capacidade de representar qualquer função, dado que a rede neural seja suficientemente complexa [Goodfellow et al., 2016]. Trabalhos provam que uma rede neural sem realimentação clássica com apenas uma camada oculta é capaz de representar qualquer função mensurável, garantindo que a rede possua quantidade suficiente de neurônios na camada oculta [Cybenko, 1989, Hornik et al., 1989]. Porém, a literatura relata que para problemas complexos, o número de neurônios em apenas uma camada oculta é proibitivamente grande, o que torna a implementação de redes com essa arquitetura impraticável. Por outro lado, o acréscimo de mais camadas ocultas permite explorar




(a) Distribuição de dados linearmente separável.

(b) Distribuição de dados tipo $x \oplus y$, não linearmente separável.

**Figura 4.4. (a) Os dados em análise podem apresentar uma distribuição que permita uma separação linear. (b) Contudo, existem dados que não são linearmente separáveis, como distribuições do tipo "XOR".**

redes neurais sem realimentação com um número menor de neurônios por camada [Goodfellow et al., 2016, Liang e Srikant, 2016]. Mesmo assim, deve haver uma ponderação sobre o número de camadas utilizadas, já que redes profundas são mais difíceis de otimizar em relação a redes rasas, devido ao número elevado de parâmetros, o que sugere que essa escolha de projeto seja feita via experimentação [Goodfellow et al., 2016].

### Redes Neurais Convolucionais

As redes neurais tradicionais, como o MLP, podem não ser indicadas para problemas com dados que apresentam muitas variáveis, além de serem ineficientes no tratamento de dados que possuam algum tipo de estrutura inerente, como relações espaciais. Para tais situações pode ser necessário um número elevado de pesos a serem ajustados, já que um neurônio se conecta às entradas de todos os neurônios da camada seguinte. Isso pode provocar um aumento no tempo de treinamento, além de exigir um número muito elevado de amostras para ajustar os parâmetros corretamente. Quando a segunda situação não é satisfeita, a rede não só não é capaz de aprender algumas variações relevantes desses dados como também fica sujeita a sobreajuste. Além disso, a estrutura de alguns dados como imagens, por exemplo, pode resultar em regiões da rede ajustadas para valores iguais [LeCun et al., 1995], o que além de um treinamento desnecessário pode causar ineficiência no armazenamento dos parâmetros da rede. As Redes Neurais Convolucionais (*Convolutional Neural Networks* – *ConvNets* – CNNs) definem estruturas que contornam tais problemas e, consequentemente, a sua aplicação já pode ser considerada uma tendência em redes de computadores. A adoção em redes é simplificada pela existência de algumas técnicas que convertem os dados provenientes das redes de computadores em formatos apropriados para o uso em CNNs [Sharma et al., 2019].

Duas características das redes neurais convolucionais se destacam: menor número de parâmetros e resistência a variações nos dados, como translação e distorção. A esparsidade de conexões entre os neurônios de camadas adjacentes e o compartilhamento de parâmetros são fundamentais para tornar mais eficiente a retirada de características locais dos dados. Dessa forma, torna-se mais difícil que apenas pedaços da rede neural se espe-

cialize em uma tarefa enquanto outros pedaços são subutilizados. A esparsidade é obtida limitando o número de conexões que um neurônio pode fazer, isto é, cada neurônio recebe apenas saídas de uma pequena vizinhança de neurônios da camada anterior [Goodfellow et al., 2016]. Um exemplo em processamento de imagens é permitir que apenas unidades de *pixels* próximos se conectem a um neurônio de uma camada posterior.

A restrição de conectividade faz com que o conjunto de neurônios passe a se especializar em identificar características primitivas dos dados analisados. Em processamento de imagens as características primitivas podem ser arestas ou contornos, por exemplo. Como essas características estão presentes em diversas partes dos dados analisados, ajustar os parâmetros para cada região implica gastos desnecessários para gerar valores similares. O compartilhamento de pesos permite que apenas um conjunto de pesos seja determinado e seja replicado para as demais regiões da camada. Na prática, apenas uma grade de valores discretos é determinada, chamada de filtro ou **Núcleo Convolucional** (*Convolutional Kernel*) [Khan et al., 2018], onde a estrutura é "passada" por todo o dado analisado. Como esperado, o tamanho dos núcleos convolucionais é menor do que a implementação das camadas como um todo, o que garante maior economia de memória e melhora na eficiência estatística [Goodfellow et al., 2016]. Os filtros convolucionais computam o produto escalar (*dot product*) entre os valores da camada anterior e os valores do filtro. O valor de cada célula dos filtros convolucionais deve ser aprendido pela rede. A representação de uma operação do filtro convolucional pode ser observada na Figura 4.5. O valor 47 presente na saída da operação é obtido a partir do produto escalar entre os elementos sombreados na matriz de entrada, região em que $i, j \leq 3$, e o filtro convolucional. O produto escalar é feito também sobre todas as submatrizes da matriz de entrada e o filtro para obter a saída completa.

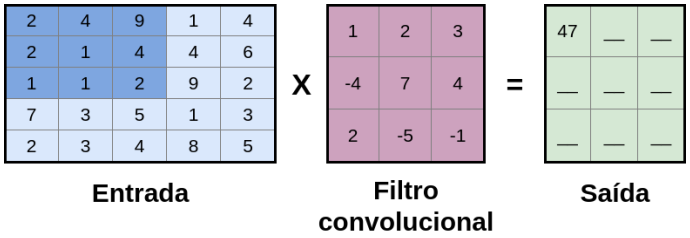



**Figura 4.5. Uma operação do filtro convolucional $3 \times 3$. A saída é obtida a partir do produto interno entre uma submatriz da matriz de entrada e o filtro convolucional. No exemplo, a submatriz é composta pelos elementos da região em que $i, j \leq 3$, formando uma matriz $3 \times 3$.**

Uma camada convolucional é tipicamente formada por três estágios. O primeiro deles, chamado de *estágio convolucional*, produz saídas lineares a partir das convoluções necessárias. O segundo estágio, chamado *estágio de detecção*, é responsável por aplicar uma função de ativação não-linear sobre os valores gerados. A saída desses estágios é chamada de mapeamento de características. O terceiro, chamado de *estágio de pooling*, opera sobre o mapeamento das características para combinar os valores em dadas regiões. Uma arquitetura típica de uma rede neural convolucional pode ser observada na Figura 4.6

As redes neurais convolucionais obtêm sua invariância quanto a pequenas translações e distorções através da função de *pooling*, como *average pooling* e *max pooling*.

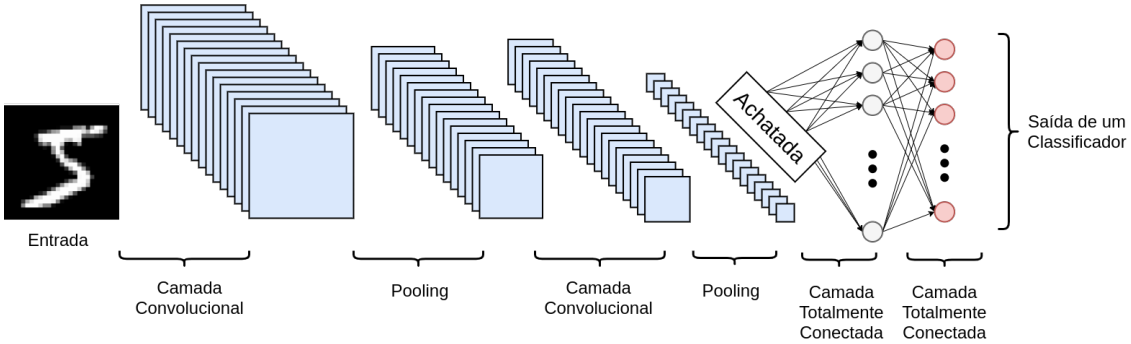



**Figura 4.6. Arquitetura típica de uma rede neural convolucional com camadas convolucionais, camadas de *pooling* e camadas totalmente conectadas.**

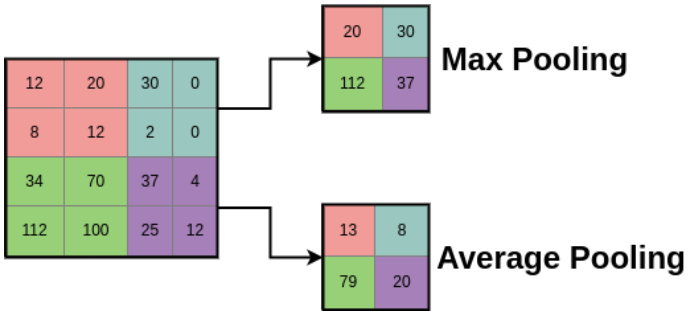



**Figura 4.7. Operações de *max pooling* e *average pooling*. Enquanto *average pooling* resulta em uma média das saídas, *max pooling* considera apenas o maior valor de uma vizinhança.**

Essas operações suavizam ou até neutralizam as variações. A operação de *average pooling* calcula a média das submatrizes de entrada, enquanto a operação de *max pooling* considera apenas o maior valor de uma dada submatriz. As duas operações estão ilustradas na Figura 4.7. Outra vantagem do uso da camada de *pooling* é a possibilidade de reduzir o número de dimensões do dado tratado, uma vez que essa camada faz uso das saídas de neurônios em uma dada vizinhança. Já a resistência das redes neurais convolucionais em relação a variações nos dados é suportada pelo uso de preenchimento por zeros (*zero padding*). Essa técnica inclui zeros nos dados de entrada da camada convolucional, aumentando a robustez da rede a dados de tamanhos variados. Isso faz com que as saídas das camadas convolucionais assumam o tamanho desejado, viabilizando o funcionamento geral da rede. Sem esse artifício o poder da rede convolucional é comprometido, visto que o desenvolvedor é forçado a escolher uma rápida diminuição na extensão espacial da rede ou o uso de núcleos convolucionais menores [Goodfellow et al., 2016].

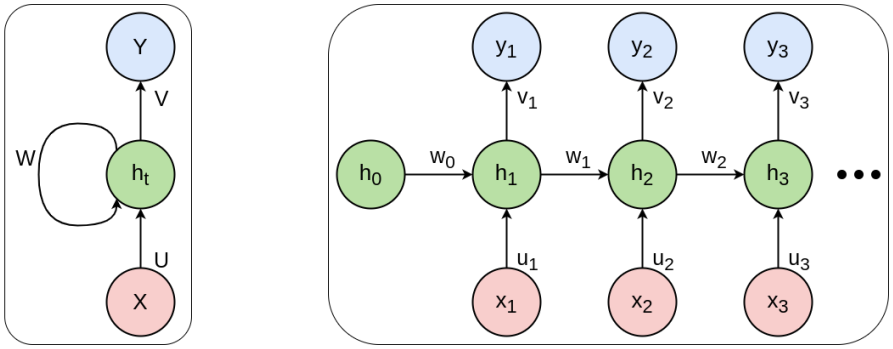
### Redes Neurais Recorrentes

Nas redes neurais sem realimentação a informação flui em um único sentido, da entrada para a saída. Essa característica torna o emprego dessas redes muito trabalhoso em cenários de tratamento de dados sequenciais. O relaxamento dessa característica, permitindo a existência de conexões cíclicas, resulta nas Redes Neurais Recorrentes (*Recurrent Neural Networks* – RNNs), ou apenas Redes Recorrentes [Graves, 2012b]. Através das conexões recorrentes, a rede tem acesso a estados anteriores, isto é, valores inseridos e

computados em iterações passadas. Isso permite que RNNs mapeiem qualquer sequência em outra sequência dado um número suficiente de unidades ocultas, conferindo um resultado equivalente à aproximação universal visto nas redes neurais sem realimentação [Hammer, 2000].

A introdução de recorrências permite que as RNNs compartilhem parâmetros entre estados através de diferentes iterações, o que permite que a rede processe sequências de tamanhos variados nas quais as informações de interesse podem ocorrer em posições variadas [Goodfellow et al., 2016]. Um exemplo são as sentenças que podem ser escritas em ordens distintas, representando o mesmo sentido. Uma rede neural sem realimentação precisa ter acesso a todas as diferentes ocorrências da sentença, enquanto uma RNN não tem essa necessidade. De modo geral, as RNNs compõem uma família de redes neurais especializada em processar dados sequenciais [Goodfellow et al., 2016]. Em um ambiente de redes de computadores, uma RNN possui maior potencial para identificar ataques que possuam características temporais ou sequenciais, como os encontrados em Ataques de Negação de Serviço (*Denial of Service* – DoS).

O compartilhamento de estados requer a introdução de novos pesos à rede para possibilitar que a saída da rede seja calculada em função das entradas, do estado da rede e das saídas anteriores. Além da diferença na computação dos resultados, também é importante mencionar que, em contraste com as entradas de tamanho fixo nas redes neurais vistas anteriormente, a entrada de uma RNN é feita de forma gradual e não possui limitação quanto ao tamanho.




(a) Representação "dobrada" de uma rede neural recorrente.

(b) Representação "desdobrada" de uma rede neural recorrente.

**Figura 4.8. Representação equivalentes de uma rede neural recorrente.**

A computação feita por uma RNN típica é dada por $\boldsymbol{h}_t = \varphi_h(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b})$ para a obtenção do novo estado e $\boldsymbol{y}_t = \varphi_y(\boldsymbol{V}\boldsymbol{h}_t + \boldsymbol{c})$ para a obtenção da saída atual, onde $\boldsymbol{b}$ e $\boldsymbol{c}$ são limiares de ativação, $\boldsymbol{U}$, $\boldsymbol{V}$ e $\boldsymbol{W}$ são, respectivamente, as matrizes de pesos das conexões da entrada para a camada oculta, da camada oculta para a saída e da camada oculta para a própria camada oculta, $\boldsymbol{h}_t$ corresponde ao estado atual, e $\boldsymbol{y}_t$ indica a saída da camada. O índice $t$ diz respeito à iteração no tratamento da sequência. Por fim, $\varphi_h$ e $\varphi_y$ são funções de ativação não-lineares. Para ilustrar as computações feitas por uma rede recorrente é comum o uso da representação desdobrada, conforme Figura 4.8(b), sendo equivalente a

representação vista na Figura 4.8(a). A representação desdobrada denota os estados da recursão como nós de uma rede, permitindo assim a análise da rede recorrente como se a rede fosse acíclica. A figura evidencia que a computação na direção da entrada para a saída (*forward*) é similar à computação executada em uma rede sem realimentação, com o acréscimo dos pesos referentes à realimentação. Outro ponto em que as duas operações se diferem é o fato de nas redes recorrentes existir a necessidade de armazenar os valores intermediários referentes a um ciclo de amostras da sequência tratável pela rede.

Uma limitação comum das redes neurais recorrentes é a dificuldade no aprendizado de dependências de longo prazo. Ou seja, as RNNs tradicionais têm dificuldade de computar a influência de estados muito distantes da iteração atual. Durante o aprendizado em redes que usam sequências longas, os gradientes tendem a desaparecer ou explodir, fenômenos chamados de *vanishing gradient* e *exploding gradient* respectivamente, inviabilizando a capacidade da rede de aprender as dependências de longo prazo [Bengio et al., 1994]. Uma forma de superar esse problema é utilizar uma variação de RNNs conhecida como Redes Recorrentes com Portas (*Gated Recurrent Networks*).

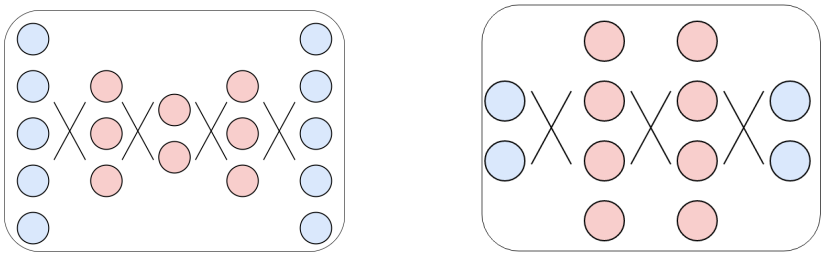
Uma das arquiteturas mais utilizadas em redes recorrentes com portas é a ***Long Short-Term Memory Network* – LSTM**. Nessa variação, as unidades computacionais da camada oculta são substituídas por uma estrutura chamada *bloco de memória*. Cada bloco é composto por uma célula de memória responsável por armazenar as informações a serem mantidas a longo prazo e pelas portas de entrada, de saída e de esquecimento (*forget gate*). As três portas são unidades de soma não-lineares, que controlam o uso da memória [Graves, 2012a]. A porta de esquecimento, quando ativada, é responsável pelo armazenamento do estado atual da unidade. As portas de entrada e saída são responsáveis pela escrita da célula de memória e pela disponibilidade da informação armazenada, respectivamente. A célula "lembra", então, a entrada de um dado instante enquanto as portas de entrada estiverem "fechadas" e a de esquecimento "aberta" [Graves, 2012a].

Em situações em que se deseja analisar uma sequência como um todo, é comum usar a arquitetura de rede chamada **Rede Recorrente Bidirecional**. Essa rede é uma composição de duas redes recorrentes unidirecionais (tradicionais) que analisam a sequência em sentidos opostos. É evidente o caráter não-causal dessas redes, indicando que essas arquiteturas só podem ser empregadas em situações onde a não-causalidade é suportada [Graves, 2012b]. Essa rede pode ser aplicada em problemas que envolvem reconhecimento de fala, nos quais a interpretação de um som em um dado estado pode ser influenciada por fonemas ou palavras que seguem ou precedem [Goodfellow et al., 2016].

### Redes Neurais Autoassociativas

O objetivo da Rede Neural Autoassociativa (*Autoencoder*) é produzir na saída uma representação que seja a mais próxima possível do dado inserido na entrada da rede. Um ponto que diferencia essa rede das outras já apresentadas é que o ajuste dos pesos da rede é feito de forma não supervisionada, uma vez que a qualidade dos pesos é verificada a partir da análise da reconstrução do dado comparado ao valor inserido anteriormente. Dividindo-a em uma estrutura codificador-decodificador, o processo de aprendizado tem como objetivo ajustar os pesos correspondentes à primeira fase, a fim de produzir uma representação codificada que possa ser recuperada na segunda fase, cujos pesos são ajus-

tados para viabilizar a reconstrução dos dados inseridos. Devido a essa estrutura, é desejável que uma rede autoassociativa consiga extrair alguma informação relevante dos dados inseridos, uma vez que uma cópia dos dados por toda a extensão da rede não seria útil para a resolução de problemas, embora essa situação satisfaça a condição exposta.



(a) Rede autoassociativa subdimensionada.

(b) Rede autoassociativa sobredimensionada.

**Figura 4.9. Exemplos de redes autoassociativas. A rede é dita (a) subdimensionada quando a dimensão das camadas ocultas são menores que a dimensão do dado de entrada. (b) Caso a dimensão das camadas ocultas seja maior do que a do dado e entrada, a rede é dita sobredimensionada.**

Usualmente uma rede autoassociativa é composta por camadas distribuídas simetricamente e respeitando o modelo codificador-decodificador [Charte et al., 2018]. Além da divisão quanto à profundidade, como presente nas demais redes (rasas e profundas), é possível caracterizar as redes autoassociativas quanto ao número de neurônios na camada intermediária. Quando a dimensão das camadas ocultas é menor que a do dado de entrada as redes são ditas Subdimensionadas (*Undercomplete*) [Goodfellow et al., 2016, Charte et al., 2018]. As redes autoassociativas subdimensionadas são comumente empregadas em problemas nos quais é desejável a redução da dimensionalidade do dado analisado, uma vez que a rede é obrigada a aprender representações mais compactas do dado inserido. Um exemplo de rede autoassociativa subdimensionada pode ser visto na Figura 4.9(a). Quando as camadas ocultas possuem dimensões maiores que a do dado inserido diz-se que a rede é Sobredimensionada (*Overcomplete*) [Goodfellow et al., 2016, Charte et al., 2018]. Essa estrutura permite a extração de mais características dos dados, produzindo uma representação em uma dimensão maior do que a do dado de entrada [Rifai et al., 2011]. A representação de uma rede autoassociativa sobredimensionada está ilustrada na Figura 4.9(b). Nessa situação, há uma preocupação adicional com o treinamento da rede, uma vez que existe uma possibilidade considerável de a rede aprender a replicar o dado de entrada por toda a extensão da rede. Para evitar esse problema é empregada uma variação da rede, chamada **Rede Autoassociativa Regularizada**, que inibe a tendência de cópias entre camadas adicionando um regularizador à função custo.

A rede **Autoassociativa Empilhada** (*Stacked Autoencoder*) é outra variação de rede neural que faz uso de redes autoassociativas. Nessa variação, constrói-se a rede neural a partir de camadas treinadas separadamente. Cada camada é tratada como uma rede autoassociativa separada, que são conectadas posteriormente, "empilhando-as" [Hinton e Salakhutdinov, 2006, Bengio et al., 2007]. Dessa forma, a implementação dessa rede consiste em um pré-treinamento não supervisionado das camadas separadamente seguido de um treinamento conjunto da rede como um todo em uma etapa de refinamento dos

parâmetros (*fine tuning*). Essas redes foram essenciais para popularizar as redes neurais profundas devido à facilidade de implementação. Elas evitam problemas associados ao treinamento de redes muito profundas, como alta variação nos parâmetros da rede, o que pode fazer com que resultados que não são ótimos sejam encontrados, além de evitar o sobreajuste. Atualmente, com o predomino do uso de aprendizado profundo em problemas de aprendizado supervisionado e com a descoberta de abordagens mais eficientes para o treinamento de redes profundas, como o *Dropout* [Srivastava, 2013], é observado um gradual abandono dessa variação de rede neural pela comunidade de aprendizado profundo [Goodfellow et al., 2016], ficando restrita a situações com pequenos conjuntos de dados [LeCun et al., 2015]. Exemplos de trabalhos recentes relacionados às Redes Desafiadoras [Lv et al., 2015, Zhang et al., 2018, Aceto et al., 2019a] ainda utilizam redes autoassociativas empilhadas. O sucesso dessas redes pode ser atribuído à presença de um grande número de amostras não classificadas nos problemas estudados. Em um trabalho para verificar se o pré-treinamento não supervisionado, como visto no treinamento das redes autoassociativas empilhadas, Paine et al. verificam que em problemas com muitas amostras não rotuladas, o ajuste da rede neural é melhorado, especialmente quando combinada a técnicas mais modernas [Paine et al., 2014]. LeCun et al. apontam para a possibilidade de um aumento de trabalhos que envolvam aprendizado não supervisionado em suas previsões para o futuro do aprendizado profundo [LeCun et al., 2015], o que pode fazer com que essa rede volte a ser mais utilizada.

Além da variante empilhada, uma outra variação de rede autoassociativa comumente utilizada é a **Autoassociativa Contrativa** (*Contractive Autoencoder*). Esse tipo de rede encoraja o aprendizado de representações mais robustas às variações nos dados de entrada. Considerando como se cada $n$ características do dado analisado estivessem representadas em um dado espaço $n$, a rede busca favorecer as direções de maior variação, que são mais significativas no dado, enquanto aquelas que são menores ou mais raras são contraídas [Rifai et al., 2011]. Forçando que os pesos sejam determinados de tal forma que $W' = W^T$, onde $W$ e $W'$ são os pesos do codificador e decodificador, respectivamente, o aprendizado da rede consegue ter um maior controle em forçar as duas estruturas a serem contrativas [Alain e Bengio, 2014]. Outra rede amplamente utilizada, a **Autoassociativa Eliminadora de Ruído** (*Denoising Autoencoder*), tem como objetivo aprender a recuperar uma representação não corrompida do dado inserido [Goodfellow et al., 2016]. Considerando $f$ e $g$ respectivamente como as funções de codificação e decodificação, pode-se dizer que a rede autoassociativa eliminadora de ruído busca robustez na reconstrução do dado, isto é, tem como objetivo encontrar a melhor $(g \circ f)(x)$, enquanto a contrativa encoraja uma melhor representação $f(x)$ [Rifai et al., 2011]. Por fim, outra rede autoassociativa regularizada popular é a **Rede Autoassociativa Esparsa** (*Sparse Autoencoder*), que é tipicamente empregada para aprender características do dado analisado para desempenhar tarefas como classificação [Goodfellow et al., 2016], usando um regularizador que força a esparsidade na codificação gerada.

## 4.3. Aprendizado Profundo Aplicado a Redes Desafiadoras

As aplicações de aprendizado de máquina costumam seguir o fluxo de trabalho ilustrado na Figura 4.10. As etapas principais são descritas a seguir:

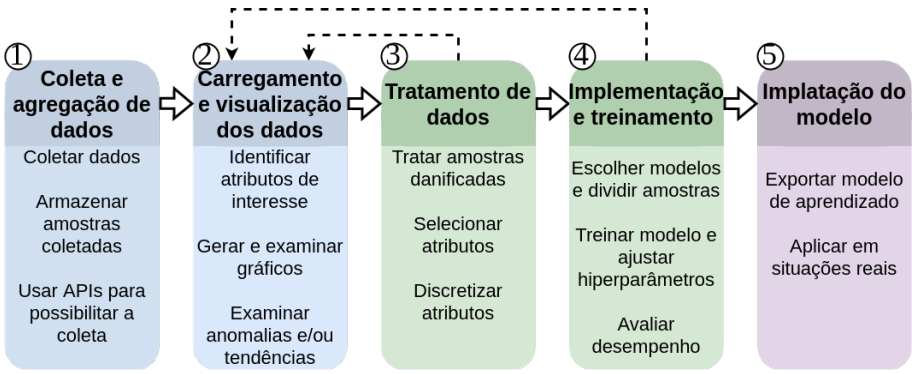



**Figura 4.10. Fluxo de trabalho típico em aplicações utilizando aprendizado de máquina.**

1. **Coleta e agregação de dados:** os dados são coletados a partir de diversas fontes, como sistemas IoT, formulários online, sítios *web* públicos, dentre outros.

2. **Carregamento e visualização dos dados:** ferramentas para geração de gráficos e análises estatísticas são usadas para analisar os dados em alto nível. Dessa forma, é possível identificar anomalias e inconsistências nos dados, além de padrões e tendências.

3. **Tratamento e pré-processamento dos dados:** com a identificação das anomalias e inconsistências, é possível tratar e pré-processar os dados que servirão de entrada para os algoritmos de aprendizado de máquina. Diversos algoritmos são usados nessa etapa e não é foco deste minicurso apresentá-los. O pré-processamento dos dados é crucial para alcançar um bom desempenho do modelo.

    - **Tratamento de amostras danificadas e formatação de atributos:** nesta etapa são tratados os dados faltantes decorrentes de um processo de coleta de dados imperfeito e valores em escalas diferentes são ajustados para uma escala única exigida por alguns algoritmos para obtenção de melhores resultados. Além disso, os atributos devem ser adequados aos algoritmos utilizados. Por exemplo, para algoritmos que recebem como entrada valores numéricos, os valores categóricos devem ser previamente convertidos. Também é necessário analisar a correlação dos atributos para possível redução em sua quantidade, ocasionando consequente melhora do tempo de treinamento e do resultado final do modelo. As transformações usadas devem ser aplicadas igualmente nos conjuntos de treino, validação e teste.
    - **Seleção de atributos com maior poder preditivo:** ferramentas estatísticas permitem identificar os atributos presentes nas amostras do conjunto de dados que são mais úteis para prever o alvo de cada amostra. Por exemplo, para a previsão da nota de um aluno é muito mais adequado utilizar a sua quantidade de horas de estudo do que a sua altura. Um algoritmo comum para composição e seleção de atributos é o de Análise de Componentes Principais (*Principal Component Analysis* – PCA), que transforma os atributos para que apenas os que possuírem maior capacidade de predição sejam selecionados.

- **Discretização (opcional):** o conjunto de dados pode ser discretizado para se adequar à entrada de um algoritmo de aprendizado de máquina. Essa discretização pode ser feita tanto de forma numérica quanto de forma categórica, dependendo do tipo de algoritmo a ser utilizado.

4. **Escolha e implementação do modelo de aprendizado:** a tarefa a ser desempenhada define o modelo ótimo para o cenário. A escolha do modelo deve considerar se o problema é supervisionado ou não supervisionado, se é um problema de classificação ou regressão, quais dispositivos são acessíveis para o treino e implantação do modelo etc.

    - **Início do treinamento:** o conjunto de dados é dividido nos conjuntos de treino, validação e teste. Após a escolha de hiperparâmetros, o modelo pode ser treinado e avaliado no conjunto de validação. Caso as métricas obtidas não sejam satisfatórias, os hiperparâmetros podem ser ajustados e uma nova rodada de treinamento pode ser executada. Apenas após a escolha final de hiperparâmetros, o modelo pode ser exportado e avaliado na etapa seguinte. A exportação do modelo significa que os pesos de uma rede neural ou os coeficientes de um regressor linear, por exemplo, estão definidos e o modelo pode ser usado em dados inéditos numa aplicação em tempo real.
    - **Avaliação dos resultados:** o conjunto de teste pode ser exposto ao modelo a fim de quantificar as métricas, como a acurácia em um problema de classificação, de forma apropriada. Ou seja, o modelo é aplicado em dados inéditos para garantir que o modelo obtém consistentemente o desempenho descrito.

5. **Implantação do modelo:** na última etapa, com o modelo treinado e com seu desempenho avaliado, é possível fazer a implantação do modelo em uma aplicação prática, seja para predição de carga, cálculo de probabilidade de falhas ou outras.

A seguir, este minicurso revisa trabalhos da literatura que aplicam nas Redes Desafiadoras o fluxo de trabalho descrito, de forma total ou parcial.

### 4.3.1. Redes Sem Fio Móveis

As comunicações através das redes sem fio vêm se popularizando, dado o crescente poder computacional dos dispositivos móveis e a consequente agregação de diferentes aplicações e serviços. Essa agregação, aliada à capacidade de mobilidade, acarreta a adesão cada vez maior dos usuários aos dispositivos móveis, provocando um aumento vertiginoso no volume de dados gerado em rede. Dessa forma, o aprendizado de máquina aplicado às **redes sem fio móveis** permite que tarefas desafiadoras como o gerenciamento dos recursos da rede, as análises em tempo real e o suporte ao crescente volume de dados sejam tratadas de forma a melhorar a experiência do usuário [Reis et al., 2020]. Dentre os trabalhos de aprendizado profundo aplicados às redes sem fio móveis, destacam-se o uso dos dados da infraestrutura ou do desempenho da rede para a previsão de informações da própria rede [Grando et al., 2019, Pierucci e Micheli, 2016] e para a classificação de tráfego [Aceto et al., 2019a, Nguyen e Armitage, 2008]. Além disso, as características da rede podem ser capturadas a partir da sua modelagem em forma de grafo, possibilitando

o estudo de seus componentes e das relações de comunicação. Essa modelagem permite determinar a importância de cada um dos nós para a operação da rede [Medeiros et al., 2017], ao identificar os nós que desempenham papéis estratégicos, como por exemplo, uma função central no controle da rede [Medeiros et al., 2016, Barbosa et al., 2019]. Esta seção discute pesquisas relevantes que aplicam o aprendizado profundo no cenário de redes sem fio móveis. O foco é dado a pesquisas relacionadas à análise de Qualidade de Experiência (*Quality of Experience* – QoE) e do tráfego de dados, que buscam aprimorar o gerenciamento ou a qualidade das redes sem fio móveis.

**Predição de QoE do usuário**

O aumento do uso dos serviços de Internet em ambientes sem fio faz com que os usuários busquem cada vez mais por serviços que ofereçam maior QoE. A resposta dos provedores é aprimorar a manutenção e operação da rede a fim de manter o nível da Qualidade de Serviço (*Quality of Service* – QoS) aceitável. Nessa direção, sabe-se que modelos de rede neural *Multilayer Perceptron* (MLP) são capazes de prever a QoE do usuário de redes móveis [Pierucci e Micheli, 2016] com base em Indicadores-Chave de Desempenho (*Key Performance Indicators* – KPIs). As estatísticas de QoS e QoE obtidas podem, por sua vez, ser usadas para guiar o desenvolvimento de novos mecanismos de controle e gerenciamento da rede utilizando, por exemplo, aprendizado por reforço [Bhattacharyya et al., 2019].

Tradicionalmente, a QoE do usuário é medida de forma subjetiva através de um grupo de voluntários participantes. No entanto, algumas características de QoS medidas a partir da operação em rede têm alta relação com a QoE do usuário e não são utilizadas como parâmetro adicional [Bhattacharyya et al., 2019]. A dificuldade de avaliar as características de QoS que podem melhorar a QoE se deve à incapacidade dos gerenciadores das redes de processar a grande quantidade de informação de QoS recebida pelo provedor de serviço. Dessa forma, o uso do aprendizado profundo se torna uma ferramenta capaz de avaliar diversas características de QoS para, então, estimar a QoE dos usuários. Nesse contexto, Pierucci e Micheli analisam uma base de dados de um provedor de serviço na Itália e destacam quais métricas de QoS medidas pelo provedor mais influenciam a QoE. A ideia é usar as métricas como entradas de um MLP a fim de melhorar a QoE [Pierucci e Micheli, 2016]. Inicialmente, divide-se o volume de dados e a vazão obtida pelos usuários em quatro regiões. Cada uma dessas regiões indica uma QoE diferente, que pode ser classificada de ruim até excelente. Os autores utilizam um MLP de duas camadas ocultas, uma com 4 neurônios e a outra com 7, e a saída da rede neural indica em qual das quatro regiões se encontra a QoE do usuário. A matriz de confusão mostra que o modelo proposto é capaz de classificar com alta precisão a QoE do usuário.

Em uma abordagem diferente, Bhattacharyya et al. focam no uso das estatísticas de QoE e QoS para melhorar o desempenho da transmissão de vídeo para os usuários. Para tanto, os autores desenvolvem a plataforma *QFlow*, baseada em aprendizado por reforço, que insere pacotes em diferentes filas de prioridade de um ponto de acesso (*Access Point* - AP). Os autores consideram um AP em situação de alta demanda trafegando dados de uma transmissão de vídeo do *YouTube*. O *YouTube* é considerado devido a sua grande quantidade de requisitos e por ocupar a maioria dos pacotes da Internet atualmente [Bhattacharyya et al., 2019]. Através de um agente controlador, a *QFlow* envia novas instru-

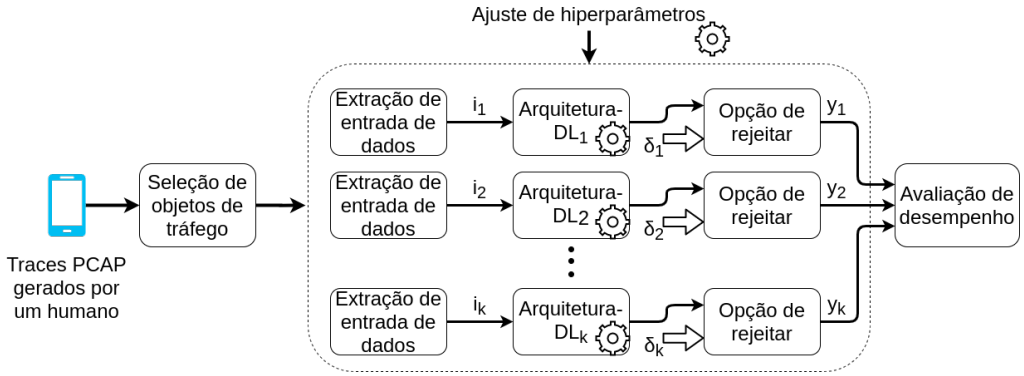



**Figura 4.11. Arquitetura de ajuste e comparação de técnicas de aprendizado, adaptado de [Aceto et al., 2019a]. Os blocos de Extração de dados de entrada selecionam uma característica do tráfego e a transforma em dados de entrada para a arquitetura de aprendizado profundo. Essa arquitetura implementa um conjunto de técnicas distintas, avaliadas quanto ao desempenho em relação à predição de tráfego gerado por aplicativos de dispositivos móveis.**

ções para atribuição de pacotes nas filas de prioridade do AP. As informações de QoE são obtidas a partir da aplicação do usuário, enquanto a QoS é obtida do AP. O agente de aprendizado por reforço usa a QoE como base para as recompensas e é constituído por uma rede neural MLP composta por duas camadas ocultas com 64 e 34 neurônios. Os autores mostram que a abordagem proposta alcança uma maior QoE quando comparada a outras abordagens de atribuições de filas como a *Vanilla* e *Round Robin*.

### Análise do Tráfego da Rede

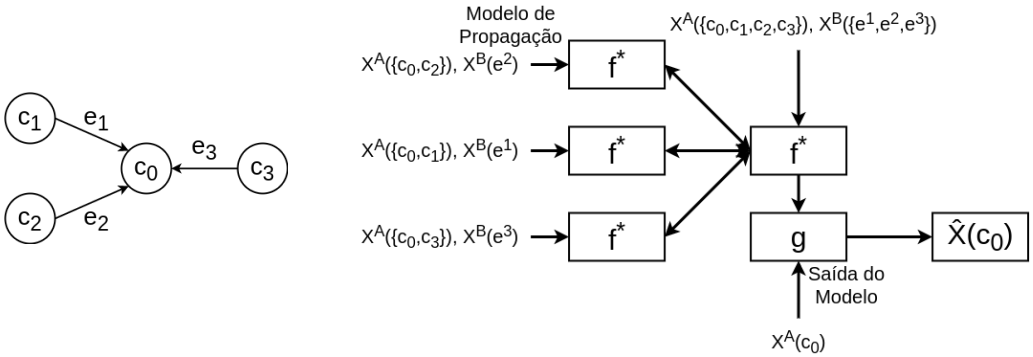
O rápido crescimento no tráfego das redes sem fio leva a estresses significativos do meio de comunicação. Com isso, há o surgimento de um grande desafio na alocação e no gerenciamento de recursos da rede, afetando diretamente a QoE dos usuários [Tang et al., 2017]. O uso do aprendizado profundo surge então como uma solução para analisar o grande volume de tráfego nas redes devido a sua capacidade de lidar com grandes quantidades de dados. Além disso, as aplicações usadas em dispositivos móveis sofrem constantes atualizações que alteram o conteúdo e a troca de dados, o que também desafia a realização tradicional da análise de tráfego. A predição do tráfego da rede pode auxiliar na alocação de recursos e, como consequência, garantir a QoS e o bom desempenho da rede [Wang et al., 2018]. Já a classificação do tráfego pode ser essencial para alguns serviços como detecção de intrusão, realocação de recursos ou a identificação dos recursos da rede utilizados pelos clientes [Nguyen e Armitage, 2008].

Aceto et al. avaliam como diferentes técnicas de aprendizado se comportam na classificação de tráfego com diferentes formas de entrada de dados. A ideia é possibilitar o uso futuro do aprendizado profundo na classificação de tráfego de dispositivos móveis [Aceto et al., 2019a]. A base de dados é um conjunto de tráfego de dados de usuários que utilizam diversas aplicações em sistemas *Android* e *iOS*. Dentre as aplicações estão o *Google Maps*, o *EFood*, o *Google Hangouts* e o *Crackle*. A Figura 4.11 ilustra a estrutura de ajuste e comparação das técnicas de aprendizado. Os dados de entrada são extraídos de diferentes formas e então aplicados a diversas técnicas de aprendizado. O desempenho

da avaliação é armazenado após analisar se a técnica é válida para os dados de entrada utilizados. A primeira e a segunda características extraídas dos dados são os primeiros bytes da carga útil do pacote e os primeiros bytes da carga total do pacote. Essas características são utilizadas como entrada para uma Rede Autoassociativa Empilhada (*Stacked Autoencoder* – SA) com cinco camadas, uma CNN com os dados de entrada em uma dimensão (1D-CNN), uma CNN com os dados de entrada em duas dimensões (2D-CNN) e uma *Long Short-Term Memory Network* (LSTM). A terceira característica extraída é baseada em campos do cabeçalho dos primeiros pacotes. Esses campos não contêm carga de dados criptografada e contêm informações como as portas de origem e destino e tamanho da janela TCP. Essas informações são usadas como dados de entrada para uma 2D-CNN, uma LSTM e um modelo híbrido no qual a saída de uma 2D-CNN é modelada como uma matriz e então utilizada como entrada de uma LSTM. Os resultados mostram que o desempenho da classificação de tráfego para o *Android* e o *iOS* é melhor quando se considera os primeiros bytes de carga útil dos pacotes e as arquiteturas que apresentam melhor desempenho são a 1D-CNN e a 2D-CNN. Vale ressaltar que todas as demais técnicas apresentam bons resultados, sendo assim, todas as técnicas são capazes de realizar a classificação do tráfego. Além disso, o estudo mostra que, apesar dos avanços, ainda é necessário maior aprofundamento para identificar uma arquitetura que sirva a todas as bases de dados com igual desempenho a partir de um modelo de entrada único. Nesse sentindo, as redes híbridas podem ser uma solução promissora.

A diversidade das aplicações e do comportamento dos usuários tornam a predição de tráfego em redes celulares desafiadora. Além disso, a mobilidade do usuário introduz uma dependência espacial, e o comportamento social, como a existência de um feriado, também influencia na demanda do tráfego da rede [Wang et al., 2018]. Apesar dessa dinamicidade e variação temporal, as Redes Neurais de Grafos (*Graph Neural Networks* – GNNs) são capazes de realizar a predição de tráfego nas redes celulares [Wang et al., 2018]. Na GNN, o conjunto de dados é modelado como um grafo para ser utilizado como entrada de uma rede neural profunda. Wang et al. usam uma GNN em um conjunto de dados composto de informações capturadas das torres da rede celular de uma cidade na China. Esse conjunto de dados possui informações importantes de cada transmissão de dados que ocorre através da rede celular, como a identificação do momento da criação do fluxo de dados, a identificação do usuário e da torre celular relacionada ao fluxo, a aplicação e o tipo de dispositivo utilizado pelo usuário. Os identificadores dos usuários são anonimizados para manter a privacidade. A informação de criação do fluxo dos dados captura a sua dependência temporal, enquanto a informação de identificação da torre captura a dependência espacial dos dados. O tráfego de dados entre os dispositivos móveis e a torre mais próxima é identificado como tráfego interno à torre ($X^A$), enquanto o tráfego entre as torres é identificado como tráfego externo à torre ($X^B$). Os dados de tráfego compõem um vetor de dados a cada período $t$ que corresponde ao *downlink* de cada meia hora. A Figura 4.12(a) ilustra a representação em grafo da recepção de dados de uma torre ($c_0$). Cada uma das torres possui um tráfego gerado pelos dispositivos ligados diretamente a ela, as arestas ($e_1$, $e_2$ e $e_3$) representam os dados que estão sendo transmitidos pelas torres vizinhas à torre $c_0$ ($c_1$, $c_2$ e $c_3$). A Figura 4.12(b) é a representação do modelo de rede neural adotado. Nessa figura, os dados de entrada são $X^A$ e $X^B$ a cada instante $t$, a função $f^*$ denota o modelo de propagação implementado por uma RNN de duas camadas ocul-

tas com 5 neurônios cada, e a função $g$ é o modelo de saída implementado por um MLP composto por duas camadas ocultas com 6 neurônios cada.




(a) Grafo da recepção de dados de uma torre celular. Em $C_i$ contém o trafégo $X^A$ e em $e_i$ contém o tráfego $X^B$.

(b) Arquitetura da técnica de aprendizado GNN. A função $f^*$ denota uma RNN e $g$ denota uma MLP.

**Figura 4.12. Modelo de predição de tráfego para uma torre celular usando GNN, adaptado de [Wang et al., 2018]. (a) O tráfego da rede é modelado como um grafo, sendo composto de tráfego entre dispositivo móvel e torre ($X^A$), e entre torres ($X^B$). (b) O tráfego serve como entrada para a arquitetura da GNN, que é composta por uma RNN e um MLP.**

Wang et al. treinam o modelo proposto e verificam seu desempenho com base no Erro Absoluto Médio (*Mean Absolute Error* – MAE). Os resultados mostram que a proposta apresenta melhor resultado quando comparada a outras redes neurais profundas, como uma LSTM. A verificação do impacto da dependência espacial é feita através da classificação das torres pela métrica de centralidade *Page Rank*, que considera as arestas ligadas à torre. As três primeiras posições mais bem classificadas pelo *Page Rank* são torres em locais de grande concentração de pessoas, como centros de compras e universidades. Por fim, os autores analisam o MAE ao longo de um dia para capturar a dependência temporal da abordagem proposta. Os resultados mostram que a proposta apresenta melhor desempenho comparada às demais redes utilizadas como comparação. O estudo realizado pelos autores mostra a importância de capturar as informações de espaço e tempo relacionadas ao tráfego para realizar a predição.

### 4.3.2. Redes IoT e de Sensores

A implementação de ambientes inteligentes que gerem maior conforto e sejam responsivos aos usuários em geral, sempre foi uma ambição da sociedade. A **Internet das Coisas** (*Internet of Things* – IoT) surge como a estrutura para alcançar essa ambição, buscando incorporar capacidade de comunicação aos dispositivos eletrônicos de forma a explorar o suporte da Internet para viabilizar a implementação dos ambientes inteligentes. Entende-se como uma rede IoT aquela formada entre dispositivos sensores e atuadores capazes de se comunicarem entre si e com a Internet [Gubbi et al., 2013]. Uma rede IoT, portanto, reutiliza conceitos tradicionais das **Redes de Sensores sem Fio** (*Wireless Sensor Networks* – WSNs) como base para o desenvolvimento de aplicações mais sofisticadas. As redes IoT empregam sensores de baixo custo econômico e energético para monitorar

um fenômeno de interesse. Dispositivos inteligentes com sensores embarcados como *smartphones* e *smartwatches* também compõem o ecossistema IoT, oferecendo aplicações personalizadas a partir da interação com servidores e outros dispositivos.

As redes IoT se caracterizam pela heterogeneidade dos dados gerados pelos dispositivos, uma vez que é habitual empregar sensores de diferentes tipos em um nó da rede. Um ponto crítico no processamento de dados em redes IoT é a qualidade dos dados gerados. Ademais, o tamanho e a disposição geográfica da rede implicam correlações espaciais e temporais entre as amostras que precisam ser levadas em consideração. Essas características podem ser exploradas utilizando técnicas de agregação de dados para, por exemplo, diminuir as transmissões na rede, aumentando consequentemente sua vida útil. Devido ao volume massivo de dados produzidos, o cenário IoT é um terreno fértil para o emprego das técnicas de aprendizado profundo. Contudo, é possível observar que o uso dessas técnicas em IoT ainda está em um estado inicial [Ma et al., 2019]. O restante desta seção apresenta algumas áreas de pesquisa relevantes para o cenário IoT. A localização passiva sem dispositivos e o reconhecimento de atividades humanas são as primeiras áreas abordadas, sendo relevantes para a implementação de sistemas inteligentes e aplicações diversas. Dessa forma, demonstra-se o uso de aprendizado profundo com o objetivo de melhorar o bem-estar do usuário. Ao final, são apresentados trabalhos que buscam melhorar a vida útil das redes, reduzindo e otimizando a quantidade de dados transmitidos.

**Localização Passiva sem Dispositivo**

A determinação da posição de pessoas e objetos em uma dada região é essencial para o desenvolvimento de muitas aplicações em sistemas IoT [Zafari et al., 2019]. Muitas das técnicas tradicionais, por exemplo aquelas que usam dados de GPS, necessitam que o objeto monitorado carregue algum dispositivo que viabilize a localização, além de geralmente exigir que o dispositivo monitorado tenha participação ativa no procedimento de localização [Youssef et al., 2007]. Como tal situação pode ser um inconveniente para algumas tarefas, uma alternativa é fazer a identificação de forma passiva, levando apenas em conta a interação do objeto com o ambiente.

Um paradigma com bastante relevância no cenário IoT, herdado das WSNs, é a Localização Passiva sem Dispositivo (*Device-Free Passive Localization* – DfP). Uma rede IoT pode ser disposta no ambiente de interesse para realizar tarefas de localização, explorando o fato de certas propriedades de sinais de radiofrequência serem alteradas em função de mudanças no ambiente [Youssef et al., 2007]. Portanto, implementar um sistema que identifique tais variações passa a ser uma alternativa interessante para localização passiva. Esses sistemas podem usar o aprendizado profundo com esse fim, sendo mais comum o uso de redes autoassociativas. Wang et al. viabilizam a identificação de dispositivos utilizando uma rede autoassociativa esparsa para identificar as mudanças no ambiente. A extração das características necessárias permite identificar a localização, a atividade e gestos executados por um indivíduo monitorado [Wang et al., 2016a]. A Figura 4.13 mostra uma configuração típica de um ambiente para identificação passiva da localização de um objeto. Os sensores formam uma rede em que cada nó pode se comunicar com os outros. A partir da medição da Intensidade do Sinal Recebido (*Received Signal Strength* – RSS) entre cada nó, a rede autoassociativa é treinada para aprender a projetar essas medições em uma dimensão menor, extraindo as características principais

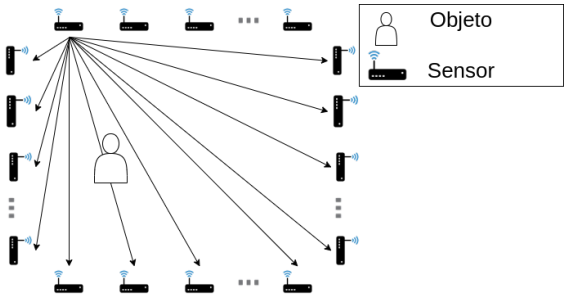



**Figura 4.13. Exemplo de um sistema de localização passiva sem dispositivo que faz uso de RSS, adaptado de [Wang et al., 2016a, Zhao et al., 2019]. Os sensores medem a RSS entre ele e os demais sensores. A RSS entre os pares de sensores é modificada de acordo com a posição do objeto no ambiente e com a movimentação desse objeto, por exemplo uma pessoa executando uma atividade.**

dos dados. Após o treinamento, a porção de decodificação da rede é descartada e a saída da nova última camada da rede autoassociativa é conectada a uma nova camada responsável por combinar os valores da saída da porção de codificação e executar a classificação. Com o acréscimo dessa camada, os parâmetros da rede são refinados, executando um novo treinamento. A abordagem baseada em aprendizado profundo difere de algumas técnicas mais tradicionais pela automação do ajuste dos parâmetros e viabilidade de identificar atividades e gestos simultaneamente, com menor sucesso na identificação de gestos.

Assim como o trabalho anterior, Zhao et al. exploram a intensidade do sinal recebido. No entanto, os autores usam uma abordagem que combina duas arquiteturas de redes neurais profundas para interpretar as variações de intensidade [Zhao et al., 2019]. Os autores usam uma rede neural autoassociativa convolucional para explorar as vantagens das redes convolucionais para o processamento de imagens e a capacidade da rede autoassociativa em ser treinada de forma não supervisionada. O sistema proposto coleta as variações percebidas por todos os sensores, armazenando os valores em uma matriz que caracteriza o estado de cada enlace entre nós da rede. Para isso, cada nó da rede transmite o sinal individualmente para os outros nós, que fazem as medições necessárias comparando-as com o valor do enlace não obstruído, isto é, o valor RSS sem um objeto.

A Figura 4.13 mostra os enlaces entre um único sensor e os demais. As medições são feitas para todos os sensores, formando uma matriz RSS. A matriz RSS obtida é convertida em uma imagem através do mapeamento de valores armazenados na matriz em *pixels*, permitindo explorar as características da parte convolucional da rede. Em seguida, a rede é pré-treinada com essas imagens, explorando a vantagem da estrutura autoassociativa da rede. As camadas correspondentes ao decodificador são compostas por camadas que realizam a operação de *max unpooling*, que consiste em desfazer as operações de convolução e *max pooling*. Após o pré-treinamento, a parte correspondente ao decodificador é descartada e substituída por uma nova camada com a mesma função vista no trabalho anterior. Após o acréscimo dessa camada, uma nova fase de treinamento é executada para o refinamento dos parâmetros.

A rede convolucional autoassociativa é comparada às redes convolucional e autoassociativa que já são empregadas nesse tipo de problema. Para valores de Relação Sinal Ruído (*Signal-to-Noise Ratio* – SNR) variando entre $-10$ dB e 15 dB, a abordagem pro-

posta pelos autores apresenta maior acurácia na localização dos objetos em todos os cenários estudados. Aliado a isso, outro ponto destacado é o curto tempo de processamento da rede quando empregada no monitoramento, sendo necessários apenas 4 ms, tornando essa abordagem extremamente atrativa para localização *online*. Vale destacar, ainda, a necessidade de apenas dezesseis sensores para obter uma acurácia adequada quando a rede é empregada em um cenário com SNR igual a 0 dB.

**Reconhecimento de Atividades Humanas**

Muitas aplicações, sobretudo de interesse médico e de segurança, têm como objetivo identificar as atividades que são executadas por um indivíduo em um determinado espaço ou período. Essas atividades podem ser observadas através de sensores externos ao indíviduo, isto é, sensores que o indivíduo não carrega consigo [Wang et al., 2016a]; ou, alternativamente, através de sensores embutidos em dispositivos utilizados pelo indivíduo monitorado [Lara e Labrador, 2012], como aqueles pertencentes a *smartphones*, *smartwatches*, dentre outros. O Reconhecimento de Atividades Humanas (*Human Activity Recognition* – HAR) é uma área importante para o desenvolvimento de soluções de cuidados de saúde inteligentes (*Smart Healthcare*) [Ma et al., 2019], em que o exame de indivíduos é feito de maneira automatizada.

Hammerla et al. investigam o uso de diferentes redes neurais no reconhecimento de atividades a partir de dispositivos vestíveis ou dispositivos fixados nos usuários [Hammerla et al., 2016]. O objetivo é extrair informações acerca das vantagens e desvantagens de cada abordagem, uma vez que é raro um estudo mais detalhado de várias redes neurais com trabalhos que usam aprendizado profundo. Estes, normalmente, apresentam apenas o desempenho dos melhores sistemas. As redes são testadas em três bases de dados. A primeira consiste em medições de indivíduos executando atividades comuns em uma cozinha. A segunda possui registros de indivíduos executando atividades predeterminadas em uma ordem variada, sendo que o objetivo do aprendizado é classificar essas atividades. Por fim, a terceira base de dados é formada por medições de indivíduos portadores de Parkinson executando atividades que induzem um problema comum da doença, que consiste na dificuldade de iniciar um dado movimento. O objetivo geral do estudo é diferenciar determinadas situações nos três cenários. Os resultados obtidos mostram que as redes LSTM apresentam um desempenho superior às CNNs na identificação de atividades de curta duração, enquanto as CNNs apresentam desempenho superior em atividades de longa duração repetitivas, como correr e andar. As redes apresentam desempenhos variados de acordo com diferentes parâmetros, porém observa-se que redes sem realimentação apresentam a maior variação, o que exige uma exploração mais aprofundada de seus hiperparâmetros para produzir resultados satisfatórios, especialmente quando comparada às demais redes. Quando as técnicas comumente empregadas são comparadas, a rede do tipo LSTM bidirecional apresenta desempenho superior na base de dados com indivíduos em atividades normais em uma cozinha, com uma margem considerável. Porém, observa-se que o número de neurônios por camada afeta consideravelmente o desempenho da rede.

Similarmente a Hammerla et al., Ravi et al. focam na classificação de atividades. No entanto, os autores utilizam acelerômetros e giroscópios embutidos em um *smartphone* e têm como objetivo desenvolver um sistema complexo o suficiente para classificações mais qualificadas das atividades sem afetar demasiadamente os recursos

disponíveis no *smartphone* [Ravi et al., 2016a]. Em um trabalho anterior, Ravi et al. observaram que a complexidade da classificação é reduzida produzindo um espectrograma dos dados de entrada [Ravi et al., 2016b]. Algumas vantagens observadas do emprego dessa técnica são a invariância quanto ao tempo e à taxa de amostragem, além de atividades com alta variabilidade apresentarem valores mais altos para várias frequências, enquanto atividades repetitivas apresentam valores altos em frequências específicas. O resultado do espectrograma serve de entrada para a rede neural. O modelo escolhido de rede neural é o sem realimentação, em que as conexões entre neurônios são limitadas de maneira similar à observada nas convolucionais. Outra decisão tomada é a de limitar o número de camadas escondidas, o que permite explorar construções hierárquicas do dado sem aumentar muito a complexidade do algoritmo. O treinamento é feito de maneira remota e a rede ajustada é carregada no celular do usuário para executar a classificação no próprio aparelho. Os resultados obtidos mostram uma acurácia superior ao modelo empregado no trabalho anterior, além de superar outros sistemas empregados para resolver o mesmo problema. Outro ponto relevante é o baixo tempo de computação encontrado, que viabiliza o reconhecimento de atividades humanas em tempo real.

Bianchi et al. focam no desenvolvimento de um sistema de monitoramento 24 h de pacientes. O objetivo é, dentro do paradigma de *Ambient Assisted Living* (AAL), identificar eventos anormais em idosos [Bianchi et al., 2019]. O sistema emprega sensores em dispositivos vestíveis que enviam os dados coletados a uma rede neural em um servidor para que as computações mais intensas sejam feitas remotamente. No trabalho, os autores comparam uma rede LSTM e uma CNN em três cenários. No primeiro cenário, as amostras são divididas de forma aleatória, respeitando 60% das amostras para o treinamento e o restante para teste. No segundo cenário, amostras de treino e teste pertencem a grupos de pessoas distintas, isto é, indivíduos presentes no conjunto de treinamento não estão no conjunto de teste. Por fim, no terceiro cenário, as amostras de todos os indivíduos estão presentes nos dois conjuntos. Constata-se que a CNN apresenta um desempenho superior ao da LSTM, principalmente no terceiro cenário. Dessa forma, os autores sugerem que para cada novo usuário, uma pequena fase de treinamento seja executada para aperfeiçoar os parâmetros da rede para o indivíduo. Comparando a rede convolucional com os métodos tradicionalmente utilizados, resultados compatíveis são observados, confirmando a viabilidade do modelo proposto.

### Técnicas para Coleta de Dados e Gestão Eficiente de Recursos

As restrições de recursos dos dispositivos IoT impõem preocupações com o desperdício de energia na transmissão de dados. Logo, é fundamental reduzir o número de transmissões para expandir a vida útil dos sensores e da rede. As técnicas de agregação e de fusão exploram características inerentes dos dados para combiná-los e podem ser usadas para diminuir a quantidade de dados a serem transmitidos. Ademais, devido à grande quantidade e à variedade nos dados gerados em redes IoT, é comum combinar as áreas de agregação e fusão com a de mineração de dados. Assim, é possível extrair informações relevantes da tarefa executada para melhorar o desempenho do sistema.

Abu Alsheikh et al. investigam o desempenho de três redes neurais autoassociativas para comprimir medições em uma rede de sensores [Abu Alsheikh et al., 2016]. A primeira rede neural é uma rede autoassociativa subdimensionada simples, capaz de obter

uma representação mais compacta do dado inserido a partir apenas de sua estrutura. Uma variação de rede autoassociativa denominada pelos autores de Autoassociativa Redutora de Pesos (*Weight Decaying Autoencoder*) também é estudada. Nessa rede, um regularizador que penaliza soluções com pesos altos nas matrizes de codificação e decodificação é acrescentado à função custo. Por fim, a terceira rede neural se trata de uma rede autoassociativa esparsa. O trabalho analisa cenários com diferentes taxas de compressão, explorando as relações espaciais ou temporais presentes nos dados. No primeiro caso, as amostras coletadas em uma dada região são analisadas em uma base central para processamento, que armazena a matriz com os pesos ajustados para codificação. Verifica-se que as redes propostas apresentam um desempenho melhor que o desempenho das técnicas tradicionalmente empregadas, como análise de componentes principais (PCA), transformada discreta de cosseno (DCT) e transformada rápida de Fourier (FFT); especialmente quando baixas taxas de compressão são utilizadas. No caso temporal, a compressão é executada no próprio sensor, a partir de amostras coletadas em um lote correspondente a um dia. Novamente, verifica-se um desempenho superior em relação às técnicas tradicionalmente utilizadas, especialmente para baixas taxas de compressão como no cenário espacial. Comparando as três redes autoassociativas, verifica-se um melhor desempenho da rede autoassociativa tradicional na reconstrução do dado comprimido. Os autores atribuem essa observação à estrutura da rede em si, acrescentando que o uso dos reguladores nas outras duas redes degradam a capacidade de reconstrução dos dados.

Em outra abordagem utilizando redes autoassociativas para compressão, Yu et al. utilizam um Veículo Aéreo Não Tripulado (VANT) como unidade de processamento da rede [Yu et al., 2018]. A Figura 4.14 ilustra o sistema proposto, em que são utilizadas redes autoassociativas eliminadoras de ruído treinadas na nuvem a partir de amostras previamente coletadas. Para explorar as características espacias das amostras coletadas, os sensores são agrupados em conjuntos com o auxilio do algoritmo *K-Means*, sendo então determinada uma rede autoassociativa para cada agrupamento. As matrizes de peso calculadas para o codificador e decodificador são armazenadas no VANT e na nuvem, respectivamente. O VANT sobrevoa a área monitorada, coleta os dados sensoreados pelos conjuntos de nós e comprime os dados antes de enviá-los para a nuvem. Por fim, com os valores referentes aos decodificadores, a nuvem é capaz de reconstruir os dados coletados. Os resultados indicam que o sistema apresenta desempenho melhor que o método baseado em Sensoriamento Compressivo (*Compressive Sensing*) ao qual é comparado. Em especial, o melhor desempenho é mais evidente em taxas de amostragens menores.

Wang et al. adotam um uso diferente do aprendizado profundo. O intuito é eliminar possíveis erros de medição provenientes de desvios nos sensores através da fusão dos dados coletados utilizando uma rede neural convolucional [Wang et al., 2017]. Erros de medição são comuns em sensores em operação por longos períodos, em que os desvios se acumulam, diferentemente do que ocorre com o ruído. A rede neural convolucional tem como objetivo extrair relações temporais e espaciais do dado coletado para eliminar os desvios nas medições. A rede é estruturada de modo que sua camada mais externa seja responsável por projetar as amostras coletadas em um espaço de características (*feature space*), enquanto as demais camadas são responsáveis por fundir esses dados a fim de eliminar os desvios. Como o tamanho do campo de recepção da camada convolucional é limitado, é desejável que dados relacionados estejam próximos na matriz de dados. Wang

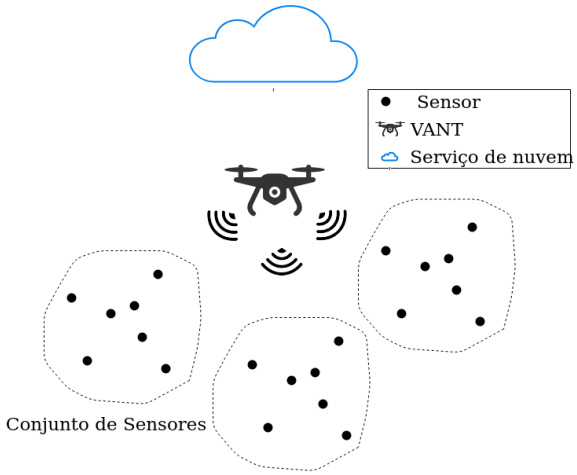



**Figura 4.14. Sistema de agregação que utiliza VANT para coletar dados de uma área e comprimir os dados utilizando um serviço de rede autoassociativa eliminadora de ruído hospedado na nuvem, adaptado de [Yu et al., 2018].**

et al. também observam a necessidade de rearranjar as amostras coletadas de tal forma que os dados vindos de sensores próximos se encontrem em regiões adjacentes na matriz, o que permite a exploração das vantagens da rede convolucional. A rede é treinada a partir de dados coletados previamente, de tal forma que os sensores são assumidos calibrados, isto é, contendo a capacidade de eliminar os desvios antes de entrarem em operação. Os autores adotam a estratégia de pré-treinar a rede inicialmente com dados que contenham pequenos desvios, seguido de um ajuste fino com dados com amostras que apresentam desvios maiores. Quando comparada a outros métodos de calibragem, a abordagem adotada apresenta uma maior taxa de recuperação, superando todos os cenários analisados. Além disso, no cenário em que menos de 15 sensores sofrem desvios, a proposta apresenta um erro menor quando a reconstrução é mal sucedida, embora apresente um erro inferior nas reconstruções bem sucedidas. Esses resultados são obtidos em comparação à técnica do trabalho anterior [Wang et al., 2016b] que usa aprendizado Bayesiano. Wang et al. realizam uma análise adicional com diferentes modelos de ocorrência de desvios. A rede convolucional tem dificuldade em encontrar pequenos desvios, impedindo a obtenção do mesmo sucesso encontrado com desvios maiores. No entanto, a rede convolucional apresenta robustez a sobreajuste dos desvios devido à forma como é treinada. De modo geral, constata-se que o uso de redes convolucionais é adequado a problemas que envolvam longos períodos de monitoramento sujeitos a desvios nas medições.

### 4.3.3. Redes Industriais

A quarta revolução industrial, ou Indústria 4.0, consiste no atual processo de modernização da indústria, sendo baseado principalmente em sistemas ciberfísicos, análise de grandes volumes de dados (*big data*) e alta dependência da Internet das Coisas Industrial (*Industrial Internet of Things* – IIoT) e de tecnologias como computação em nuvem e em névoa [Aceto et al., 2019b]. O conceito de IIoT é um caso específico de IoT, que conecta os elementos industriais, como máquinas e sistemas de controle, com sistemas de informação e processos de negócios. Como consequência, um grande volume de dados é coletado para que soluções analíticas sejam implementadas e operações industriais sejam

otimizadas. Entre as principais diferenças entre IIoT e IoT estão o volume superior de dados no ambiente industrial e algumas diferenças em requisitos de comunicação como QoS, disponibilidade, segurança e privacidade. As tecnologias sem fio são tipicamente adotadas em IIoT para as aplicações industriais, as quais são capazes de alcançar grande flexibilidade e escalabilidade [Sisinni et al., 2018]. Assim, na Indústria 4.0, os sistemas de fabricação são capazes de monitorar os processos físicos e tomar decisões inteligentes através de comunicações em tempo real e interação com humanos, máquinas, sensores etc. Essa modernização permite que os processos de fabricação sejam reconfiguráveis, dinâmicos e flexíveis a fim de atender um mercado global e dinâmico [Zhong et al., 2017].

A Indústria 4.0 se relaciona a três conceitos-chave: produção inteligente, produção baseada em IoT e produção em nuvem [Zhong et al., 2017]. Na produção inteligente as decisões são baseadas em inteligência artificial e o processamento de grandes massas de dados (*big data*) é uma de suas tecnologias de suporte [Costa et al., 2012]. Na produção baseada em IoT, os dados provenientes de sensores precisam ser coletados em tempo real, assim como os processos de produção precisam ser visíveis e rastreáveis também em tempo real. Dessa forma, a tomada de decisão que afeta a produção é realizada com o menor atraso possível. Por fim, na produção em nuvem, os serviços podem ser distribuídos e compartilhados. Enquanto as informações são coletadas em toda fábrica, estas podem ser analisadas na nuvem e casos como defeitos na linha de produção ou temperaturas acima do padrão estabelecido podem ser detectados sem investir em equipamentos físicos ou utilizar poder computacional além do necessário para a execução da tarefa. Existem na literatura propostas que buscam aprimorar a produção utilizando redes neurais profundas. Por exemplo, a análise da confiança das aplicações sem fio na produção é desafiadora, dada a quantidade de dados a serem analisados. Ao mesmo tempo, é de grande importância, uma vez que as redes sem fio industriais são adotadas na IIoT. Nesse contexto, Sun e Willmann desenvolvem um sistema para avaliação de confiança [Sun e Willmann, 2019]. Já Zhang et. al propõem um sistema capaz de predizer a carga de trabalho na nuvem em intervalos futuros unindo a produção inteligente com o conceito de produção em nuvem [Zhang et al., 2018]. Li et al. propõem um sistema que une produção inteligente, produção baseada em IoT e produção em nuvem capaz de avaliar defeitos na produção com o auxílio de nós em névoa ou na nuvem a partir de dados coletados por diversas câmeras na linha de produção [Li et al., 2018].

### Avaliação de Confiança em Redes Industriais Sem Fio

A confiança é um métrica que pode ser quantificada através de vários parâmetros. Sun e Willmann propõem um método para auxiliar a avaliação da confiança em aplicações sem fio industriais. A proposta foca na redução da dimensão dos parâmetros antes da fase de treinamento de um modelo de aprendizado. Dessa forma, tornam-se mais fáceis o processo de agrupamento das características das redes industriais e, consequentemente, a rotulação que auxilia a avaliação de confiança das aplicações [Sun e Willmann, 2019]. O modelo proposto pelos autores pode ser utilizado em diversos cenários envolvendo sistemas de comunicação sem fio, como sistemas de manutenção preditiva ou robôs industriais. Os dados de entrada são parâmetros de qualidade que podem ser mensurados, como o Tempo de Transmissão (*Transmission Time* – TT), ou calculados, como a Taxa de Mensagens Perdidas (*Lost Message Ratio* – LMR), por exemplo. O funcionamento

do sistema em um cenário genérico pode ser observado na Figura 4.15, onde os dados são inicializados, e também normalizados, antes mesmo do treinamento. A inicialização considera fatores de tempo, já que a confiança da rede muda dinamicamente. Esses dados são comprimidos pela rede profunda autoassociativa na fase de treinamento. Vale notar que após o treinamento, os dados passam apenas pela parte codificadora da rede autoassociativa antes que o agrupamento seja executado. O Algoritmo DBSCAN (*Density-Based Spatial Clustering of Applications with Noise*) é responsável por realizar o agrupamento. Os grupos são usados para calcular pontuações de confiança. O modelo é capaz de realizar o agrupamento e mostrar sua relevância uma vez que a quantidade de parâmetros para avaliar a confiança em aplicações sem fio industriais aumenta conforme o crescimento da complexidade das aplicações, tornando a avaliação da confiança mais difícil sem o auxílio de mecanismos como o proposto.

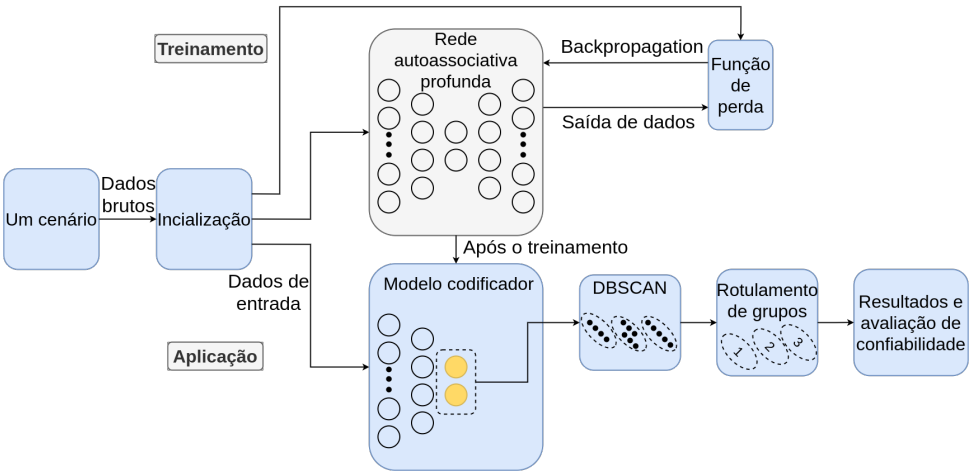



**Figura 4.15. Processo para avaliação inteligente de confiança de aplicações sem fio, adaptado de [Sun e Willmann, 2019].**

### Predição da Carga de Trabalho na Nuvem

A computação em nuvem é uma das tecnologias empregada na modernização industrial por prover recursos computacionais e serviços sob demanda, além de possibilitar o armazenamento e processamento dos dados com baixo custo. Por ser uma das principais habilitadoras da Indústria 4.0, é importante avaliar e predizer a carga de trabalho na nuvem imposta pelas máquinas industriais. A predição da carga de trabalho nesse cenário é desafiadora, uma vez que as máquinas geram cargas de trabalho de forma dinâmica. A predição nesse caso permite que a qualidade dos serviços seja garantida e que os recursos da rede industrial sejam otimizados. Zhang et al. apontam que o treinamento de um modelo profundo é uma tarefa com alto consumo de tempo dado o grande número de parâmetros envolvidos e propõem um modelo para predição de carga de trabalho na nuvem adotando Decomposição Poliádica Canônica (*Canonical Polyadic Decomposition*) para comprimir os parâmetros [Zhang et al., 2018]. O modelo tem como objetivo predizer a utilização de CPU da máquina virtual com maior carga de trabalho em um dia futuro e a utilização de diversas máquinas virtuais em quatro intervalos futuros. A proposta busca também reduzir o tempo de execução e lidar com a grande quantidade de parâmetros.

A avaliação do treinamento é realizada através de quatro métricas: erro de apro-

ximação, queda de acurácia da classificação, redução de parâmetros e aumento da velocidade de treinamento. O erro de aproximação e a queda de acurácia são causados pela conversão de parâmetros. A redução dos parâmetros é a taxa entre o número de parâmetros originais e comprimidos. Já o aumento de velocidade de treinamento refere-se à taxa entre o tempo de execução do treinamento tradicional das redes autoassociativas empilhadas com o do modelo proposto. O modelo proposto é comparado com o tradicional e com outro modelo que realiza a compressão de parâmetros através de outro método, conhecido como *Tucker decomposition*. Para avaliar a acurácia de predição, a comparação é feita com as técnicas tradicionais de redes neurais e de *Deep Belief Networks* (DBNs). O modelo proposto pelos autores alcança velocidade de treinamento superior aos demais com baixa perda de acurácia. A predição de utilização de CPU com maior carga de trabalho é realizada com maior acurácia que as outras técnicas de aprendizado de máquina.

**Inspeção de Produtos através de Classificadores**

Ao unir os conceitos de IIoT e aprendizado profundo, e utilizar a computação na nuvem, Li et al. desenvolvem um sistema capaz de identificar e classificar defeitos na linha de produção de forma inteligente, tendo a sua fabricação monitorada por diversas câmeras. Dessa forma, o sistema busca operar em tempo real, apesar da grande quantidade de dados gerada pelas câmeras. Para alcançar seus objetivos, os autores utilizam nós em névoa em conjunto com nós na nuvem para reduzir o tempo de execução [Li et al., 2018]. Os autores ressaltam que as soluções baseadas em redes neurais profundas melhoram a análise e reconhecimento de padrões para a detecção de defeitos na linha de produção, mas possuem o poder computacional como empecilho para implementação de um sistema em tempo real. Uma das possibilidades para solucionar esse problema é implementar as redes neurais profundas em servidores na nuvem, mas essa abordagem completamente *online* pode aumentar o tempo de execução. Dessa forma, o sistema proposto adota também a computação *offline* através de nós em névoa.

O sistema proposto por Li et al. identifica defeitos nos produtos e mensura o grau de gravidade, classificando-os como dentro ou não das conformidades de acordo com as políticas da fábrica. A Figura 4.16 mostra o funcionamento do modelo proposto. Inicialmente as imagens captadas por câmeras na linha de produção são enviadas aos computadores locais, ou seja, os nós da névoa. Em seguida, os dados passam por duas camadas de redes convolucionais que estão nos nós locais para tentar classificar os defeitos dos produtos. Os dados resultantes da classificação realizada pelos nós locais são chamados de resultados intermediários. Paralelamente, os resultados são enviados para os servidores na nuvem, passando por mais duas camadas convolucionais e duas camadas completamente conectadas, sendo possível realizar tanto a classificação dos defeitos quanto medir o grau de comprometimento através de um regressor. O resultado das camadas dos nós em névoa é considerado como final caso a abordagem *offline* seja capaz de analisar corretamente as imagens da câmera. Então os resultados intermediários deixam de ser enviados à nuvem. O sistema determina se a abordagem *offline* é ou não capaz de realizar a análise utilizando um limite definido como o máximo aceitável para a função de custo. Esse resultado final obtido sem os servidores na nuvem é chamado de *early exit*, uma vez que o sistema realiza a avaliação de defeitos mais rapidamente. É possível adotar mais de uma *early exit* na abordagem *offline*, enviando os dados paralelamente não apenas aos servidores na nuvem,

mas também para outras camadas convolucionais. Uma camada de *extra max pooling* é utilizada para reduzir o tamanho dos dados que são enviados aos servidores.

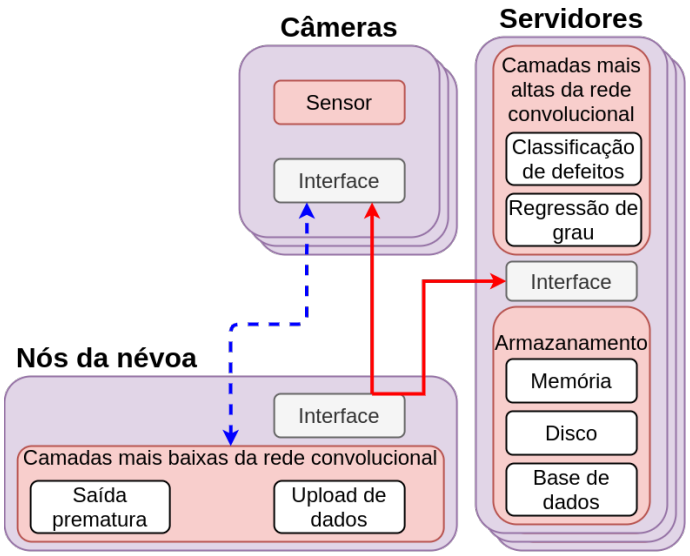



**Figura 4.16. Sistema para avaliação de defeitos na linha de produção, adaptado de [Li et al., 2018].**

Um dos desafios enfrentados pelos autores é o desenvolvimento de uma função de custo que permita o treinamento eficiente do regressor e do classificador simultaneamente. A função de custo *online* proposta é a soma de três outras funções e seus respectivos pesos com objetivos distintos: identificar o defeito através de uma função de custo baseada na função *softmax*, mensurar o grau de comprometimento do defeito e reduzir o sobreajuste. A função de custo da proposta *offline* é o somatório das funções de custo de cada saída prematura (*early exit*), sendo essas iguais à função de custo da proposta *online*. O conjunto de dados capturado manualmente para testar o sistema é composto por dez categorias de defeitos distintos. A habilidade de diagnóstico do sistema de classificação é comparada com dois outros métodos utilizados na detecção de defeitos, a detecção de contornos (*contour detection approach*) e a detecção baseada em pixels (*pixel-based method*), através da curva Característica de Operação do Receptor (*Receiver Operating Characteristic* – ROC). Em pelo menos nove situações o sistema dos autores possui melhor desempenho e reduz o tempo de execução em relação à computação local.

### 4.3.4. Redes Veiculares

As informações veiculares e do fluxo de veículos nas vias permitem aos sistemas de transporte inteligentes (*Intelligent Transportation Systems* – ITS) diversas análises para proporcionar melhor qualidade, conforto e segurança nas viagens. Nesse sentido, o aprendizado profundo pode ser adotado em redes veiculares para aprimorar o aprendizado da dinâmica da rede e a alocação e o gerenciamento de recursos. O aprendizado da dinâmica da rede pode envolver grandes quantidades de dados para que seja possível o desenvolvimento de sistemas inteligentes capazes de aprimorar a predição do fluxo de veículos ou a precisão da análise de segurança nas estradas [Lv et al., 2015, Peng et al., 2018]. Além disso, as redes neurais profundas são adotadas também para alocar recursos na co-

municação entre veículos (*Vehicle-to-Vehicle* – V2V), para permitir que a potência para transmissão e sub-banda sejam otimizadas sem a necessidade de requisitar ou aguardar informações globais do sistema [Ye et al., 2019].

### Aprendizado da Dinâmica da Rede

Os sistemas de transporte inteligentes viabilizam diversas análises sobre o tráfego veicular, que permitem definir a trajetória de uma viagem de forma a reduzir os congestionamentos e a emissão de gás carbônico [Lv et al., 2015]. Além disso, as informações sobre o tráfego veicular permitem avaliar e buscar formas de aumentar a segurança dos condutores e pedestres nas estradas [Peng et al., 2018]. Nesse contexto, Lv et al. propõem um modelo de aprendizado profundo utilizando redes autoassociativas empilhadas para que as características genéricas do fluxo de tráfego sejam aprendidas e o comportamento futuro possa ser predito [Lv et al., 2015]. Após a rede autoassociativa empilhada, a arquitetura emprega um preditor na camada de saída. O modelo proposto é realizado para fazer a predição do fluxo de tráfego em quatro intervalos futuros: 15 minutos, 30 minutos, 45 minutos e 60 minutos. Em cada um dos intervalos pretendidos é utilizado um número diferente de neurônios por camadas, determinado a partir de testes experimentais.

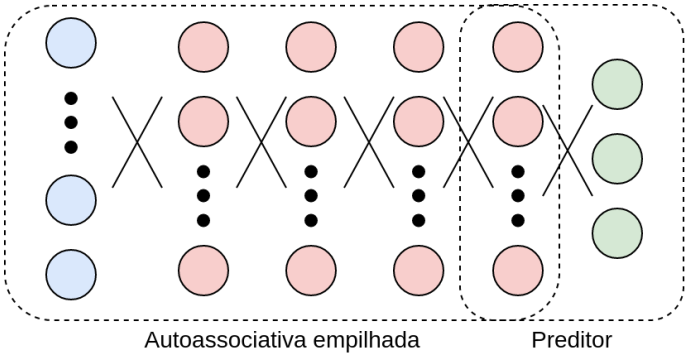



**Figura 4.17. Arquitetura do modelo de predição do fluxo de tráfego, adaptado de [Lv et al., 2015].**

A Figura 4.17 apresenta a arquitetura do modelo proposto pelos autores. Inicialmente, as características dos dados de entrada são extraídas pela rede autoassociativa empilhada antes de passar pelo preditor. Os dados que formam o conjunto de dados de entrada são coletados periodicamente por diversos detectores distribuídos pelos sistemas rodoviários da Califórnia. Antes de serem submetidos à rede autoassociativa empilhada, as informações obtidas por cada detector individualmente são agregadas em intervalos maiores. Em rodovias com múltiplos detectores, os dados de tráfego coletados pelos diferentes sensores são agregados para gerar o fluxo médio da rodovia. Na fase de treinamento do modelo, as camadas são treinadas das mais próximas da entrada para a saída conforme a Figura 4.17. O objetivo é minimizar a função de custo antes de treinar a próxima camada com a saída da camada anterior. A saída da última camada da rede autoassociativa empilhada é usada como entrada da camada de predição. O preditor inicializa seus parâmetros aleatoriamente ou através de treinamento supervisionado. Por fim, os parâmetros são ajustados em todas as camadas utilizando *backpropagation* de forma supervisionada.

Os autores avaliam o desempenho da proposta a partir de três indicadores: Erro

Médio Absoluto (*Mean Absolute Error* – MAE), Erro Médio Relativo (*Mean Relative Error* – MRE) e Erro Quadrático Médio (*Mean Squared Error* – MSE). O modelo é analisado de forma comparativa com outras abordagens para predição de tráfego, como redes neurais com Função de Base Radial *(Radial Basis Function* – RBF). A proposta dos autores alcança acurácia média superior aos outros modelos comparados. O melhor desempenho obtido pela proposta dos autores é, em parte, resultado do ajuste adequado de hiperparâmetros, como o número de camadas escondidas e a quantidade de neurônios em cada camada. Dessa forma, o experimento comprova que o dimensionamento correto dos hiperparâmetros é importante para obter os melhores resultados possíveis.

Enquanto Lv et al. focam na definição de trajetórias, Peng et al. usam uma rede neural híbrida para aumentar a segurança de condutores e pedestres [Peng et al., 2018]. As pesquisas sobre a segurança veicular geralmente são feitas através de duas abordagens distintas. Na primeira, informações veiculares para análise de segurança dos condutores e pedestres são utilizadas; enquanto na segunda, imagens e fatores externos para analisar a segurança das estradas são consideradas. No entanto, as abordagens para análise da segurança nas estradas com base em fatores externos que utilizam modelos matemáticos podem não refletir bons resultados em todos os ambientes urbanos por assumir dados e parâmetros empíricos. A análise das imagens não possui grande precisão por focar apenas em imagens locais. Os autores tentam minimizar esses problemas propondo o *deep learning framework* (DeepRSI), uma rede neural híbrida que melhora a precisão da análise de segurança nas estradas, através da captura tanto das informações contidas nos sensores de carros e dispositivos, quanto de fatores externos do ambiente, como o clima [Peng et al., 2018]. O conjunto de dados de entrada para o modelo é um conjunto de dados urbanos da cidade de Nova Iorque, que inclui dados climáticos, eventos, e dados coletados por 13 mil táxis que apresentam as rotas por GPS e fornecem informações dos sensores dos veículos.

A Figura 4.18 ilustra o arcabouço DeepRSI composto por três redes convolucionais e uma rede totalmente conectada (*Fully Connected* – FCs) para cada região da cidade dividida em malha. O aprendizado espaço-temporal é atribuído às redes convolucionais. Para tanto, o conjunto de dados é dividido nos segmentos temporais "recentes", "próximos" e "distantes". Cada segmento é enviado a uma rede convolucional distinta, que realiza o aprendizado espacial a partir das camadas convolucionais densas. As saídas são então agregadas através de um método de fusão baseado em matriz paramétrica. A parte completamente conectada recebe como dados de entrada os fatores externos do ambiente e sua saída é adicionada às saídas convolucionais. A função de ativação ReLu (*Rectified Linear Unit*) tem como objetivo tornar a convergência mais rápida e, por fim, a função de ativação *softmax* faz a classificação para a saída. A saída é o índice de segurança (*Safety Index* – SI) de cada região. Assim, o SI permite avaliar a segurança das rodovias levando em consideração todos os aspectos importantes, ou seja, tanto as informações relacionadas aos veículos e pedestres, quanto as informações relacionadas ao ambiente.

A avaliação do DeepRSI é feita através da análise das métricas de **precisão** e **sensibilidade (*recall*)**. Os resultados mostram que o DeepRSI apresenta melhor desempenho quando comparado a outras abordagens como Árvore de Decisão (*Decision Tree* – DT) e Máquina de Vetor de Suporte (*Support Vector Machine* – SVM). Os autores defendem que o arcabouço proposto é suficientemente genérico para ser aplicado em qualquer ce-

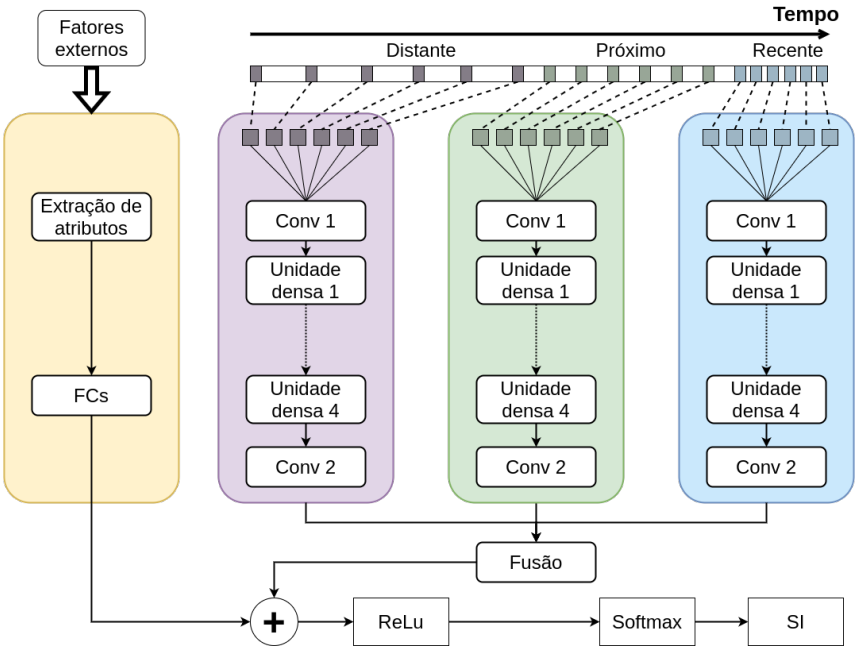



**Figura 4.18. DeepRSI é um arcabouço que usa uma arquitetura híbrida para avaliar a segurança em estradas através do seu índice de segurança (SI), adaptado de [Peng et al., 2018]. As redes convolucionais recebem como entrada os dados relacionado ao veículos, e a rede completamente conectada recebe as informações relacionadas a fatores externos, como o clima.**

nário, pois seu aprendizado não está limitado a imagens locais somente, como em outras abordagens relacionadas à segurança nas estradas. O fator mais importante do DeepRSI é a agregação dos dados geralmente utilizados para a segurança de condutores e pedestre para análise de segurança nas estradas, o que implica em maior refinamento da análise.

### Alocação e Gerenciamento de Recursos

As comunicações nas redes veiculares podem ocorrer fundamentalmente entre veículos (V2V) ou entre veículo e infraestrutura (*Vehicle-to-Infrastructure* – V2I). Mais recentemente, porém, surgiu o paradigma V2X (*Vehicle-to-Everything*), no qual a comunicação ocorre entre veículos e outros sistemas de comunicação como as redes celulares. Independentemente do cenário, a comunicação veicular é essencial para reforçar a segurança nas estradas. No entanto, essa comunicação é complexa e sofre com diversas interferências inerentes ao ambiente de comunicação sem fio. A alocação de recursos é uma abordagem comumente utilizada em redes sem fio de uma forma geral para melhorar a comunicação entre os dispositivos. Nas redes veiculares, alguns autores propõem o uso de modelos de aprendizado profundo para promover uma alocação de recursos mais eficiente. Normalmente, os mecanismos de alocação de recursos para comunicação V2V são centralizados, recaindo em um problema de otimização NP-difícil. Além disso, a centralização do mecanismo pode levar a uma sobrecarga de controle na rede. Para evitar esses problemas, pode-se utilizar uma abordagem descentralizada, que tende a ser mais autônoma e robusta. Nesse contexto, Ye et al. propõem um sistema descentralizado baseado em aprendizado profundo por reforço para promover a comunicação eficiente em uma VANET (*Vehicular Adhoc NETwork*) [Ye et al., 2019]. Os autores implementam um

agente autônomo capaz de tomar decisões e determinar a sub-banda e a potência de transmissão que otimizam a comunicação no momento. Isso é feito sem que o agente requisite informações globais.

A proposta de Ye et al. usa uma função de recompensa que destaca o contraste entre a comunicação V2V e a V2I. O objetivo é garantir os requisitos de latência do V2V e, ao mesmo tempo, garantir que a comunicação V2V não interfira na comunicação V2I. O modelo proposto usa *Deep Q-Learning* para alocar os recursos na rede. A proposta pode ser aplicada em cenários *unicast* ou *broadcast*. No cenário *unicast*, as mensagens frequentemente não alcançam todos os veículos em uma vizinhança, já no cenário *broadcast* elas podem prejudicar a comunicação devido ao excesso de colisões de pacotes. Aplicando o modelo proposto ao cenário *unicast* a cada instante de tempo $t$, o agente observa o estado da comunicação e, de acordo com a política de aprendizado, realiza a ação de selecionar uma sub-banda e a potência da transmissão. Já no cenário *broadcast*, além das ações para a transmissão, o agente também retransmite uma mensagem que ele próprio recebeu anteriormente por *broadcast*, dentro de um subconjunto de mensagens existentes.

Os treinos e testes do modelo proposto são feitos para um ambiente simulado. Os veículos são dispostos na via aleatoriamente e supõe-se que cada veículo precisa estabelecer uma comunicação com outros três veículos. A quantidade de veículos varia de 20 a 160 durante o experimento. A arquitetura do modelo usa uma rede neural completamente conectada com cinco camadas, sendo três camadas ocultas com 500, 250 e 120 neurônios. O modelo proposto é comparado a outros métodos em ambos os cenários *unicast* e *broadcast*, e apresenta melhor aproveitamento dos enlaces de comunicação. No entanto, o modelo proposto apresenta elevado tempo de computação devido à complexidade, sendo necessário promover melhorias para reduzir esse tempo e viabilizar seu uso.

### 4.3.5. Segurança nas Redes Desafiadoras

A grande quantidade de dispositivos existentes nas Redes Desafiadoras com capacidade de comunicação e acesso à Internet aumenta significativamente a superfície de ataque da rede. Se não existirem mecanismos de proteção eficientes, os dispositivos nessas redes constituem uma porta de entrada para usuários maliciosos. Em redes industriais, por exemplo, a proteção de sistemas de controle conectados à Internet é primordial para evitar novas formas de ataque às estruturas críticas. Nas Redes Desafiadoras, os Sistemas de Detecção de Intrusão (*Intrusion Detection Systems* – IDSs) são uma necessidade de segurança básica comum a todos os paradigmas e são factíveis utilizando aprendizado profundo. A vantagem dos IDSs baseados em aprendizado profundo sobre aqueles que utilizam técnicas clássicas de aprendizado de máquina é a redução na taxa de falsos positivos e a menor dificuldade em identificar novos tipos de ataques [Al-Hawawreh et al., 2018]. Esta seção discute os pontos mais importantes no desenvolvimento de sistemas de segurança nas Redes Desafiadoras, porquanto muitas das decisões tomadas ao abordar um tipo de rede permeiam as outras redes discutidas.

Al-Hawawreh et al. propõem um sistema de detecção de anomalias cuja arquitetura é baseada em redes autoassociativas e redes neurais sem realimentação. Os conjuntos de dados utilizados na avaliação da proposta são compostos por dados coletados a partir de

tráfego TCP/IP, divididos em três subconjuntos. Enquanto o subconjunto A apenas possui amostras do comportamento normal da rede, os subconjuntos B e C possuem amostras normais e de ataques [Al-Hawawreh et al., 2018].

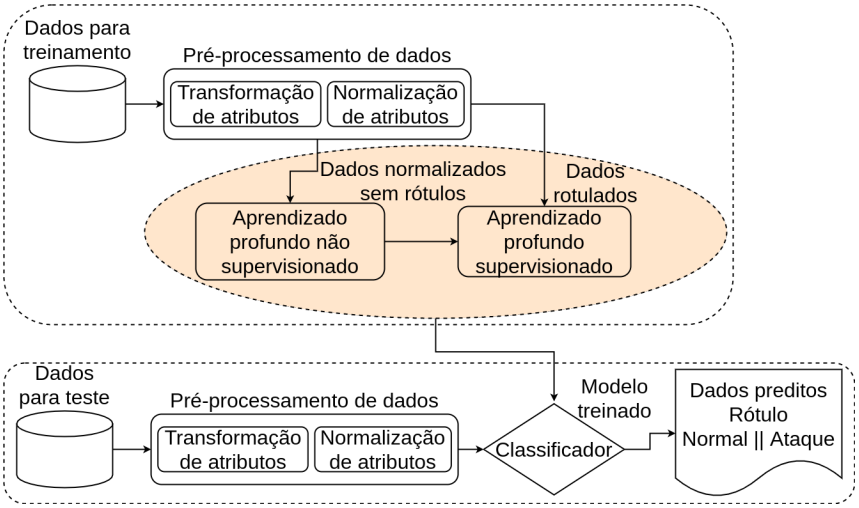



**Figura 4.19. Arquitetura do sistema de detecção de anomalias em redes industriais, adaptado de [Al-Hawawreh et al., 2018].**

A Figura 4.19 descreve o funcionamento do sistema proposto, onde inicialmente os dados são transformados e normalizados antes de iniciar a fase de treinamento e teste. Na transformação, os valores não numéricos são mapeados em valores numéricos e a normalização adotada é a *Z-score*, que mapeia os valores em torno da média do conjunto. O treinamento é realizado em duas fases. O objetivo da divisão do treinamento é fazer com que o algoritmo responsável pela classificação seja inicializado com pesos e limiares de ativação (*biases*) ótimos, ao invés de inicializar esses parâmetros com valores aleatórios. A aleatoriedade pode incorrer em mais tempo necessário para a convergência do algoritmo. A primeira etapa do treinamento ocorre com o auxílio de redes profundas autoassociativas utilizando o subconjunto de dados A. Na segunda fase, a acurácia do método é testada com as amostras do subconjunto B, que inclui dados de ataque. O erro médio quadrático é definido como a função de custo a ser minimizada na fase de treinamento, utilizando o gradiente descendente estocástico. Ao final dessa etapa o modelo para predição está pronto para ser testado. Por fim, o subconjunto C é utilizado na fase de testes. Os autores avaliam o sistema usando os conjuntos de dados NSL-KDD e UNSW-NB15. Os resultados mostram que a proposta alcança melhor desempenho no cenário NSL-KDD, obtendo 98.6% de acurácia, 99% de taxa de detecção e apenas 1.8% de taxa de falsos positivos. No UWSN-NB15, a proposta também apresenta bom desempenho, com acurácia de 92.4%, taxa de detecção de 93% e 8.2% de taxa de falsos positivos.

Diferentemente do trabalho anterior, Sharafaldin et al. investigam o desempenho de uma técnica de aprendizado profundo comparada a modelos tradicionais de aprendizado de máquina para a detecção de anomalias [Sharafaldin et al., 2018]. A abordagem dos autores implementa o modelo MLP. Os autores utilizam também um modelo de regressor de floresta aleatória (*random forest regressor*) para extrair o melhor conjunto de atributos dentre os 78 atributos originais de um conjunto de dados. Dessa forma, é possível inferir quais atributos possuem maior **poder de predição** em relação ao alvo. Os

atributos escolhidos variam de acordo com o tipo de ataque realizado na rede. É interessante ressaltar que a etapa de pré-processamento indica os atributos mais efetivos para a identificação e classificação de ataques, mesmo sem a aplicação do modelo profundo. Na segunda etapa, os autores utilizam seis modelos de aprendizado de máquina tradicional e o modelo MLP de aprendizado profundo. Os modelos são avaliados segundo as métricas de precisão, sensibilidade (*recall*) e **pontuação F1** (*F1-Score*). Os resultados mostram que o modelo de aprendizado profundo proposto não atinge um desempenho satisfatório.

Analogamente ao trabalho anterior, Panwar et al. também utilizam modelos tradicionais de aprendizado, evidenciando a necessidade de exploração de modelos profundos e seu desempenho para o desenvolvimento de sistemas de detecção de anomalias [Panwar e Shailesh, 2019]. No entanto, apesar dos trabalhos de Sharafaldin et al. e Panwar et al. mostrarem que os modelos de detecção de anomalia baseados em técnicas de aprendizado clássicas apresentam melhor desempenho quando comparadas aos modelos baseados em aprendizado profundo, Vinayakumar et al. apresentam uma análise detalhada que evidencia o potencial de redes neurais profundas para a classificação de ataques em diferentes tipos de redes de computadores [Vinayakumar et al., 2019]. Além de uma análise extensiva sobre os conjuntos de dados NSL-KDD, UNSW-NB15 e CICIDS, os autores também realizam uma análise sobre o ajuste de hiperparâmetros do modelo de aprendizado profundo. Os autores alcançam uma acurácia de $96,2\%$ para classificação multiclasse e $96,3\%$ de acurácia para classificação no conjunto de dados CICIDS com redes neurais profundas regularizadas de 3 camadas e 1 camada. O melhor desempenho alcançado com modelos de aprendizado clássico é obtido com o modelo de floresta aleatória (*random forest*), com $94,4\%$ de acurácia na classificação multiclasse e $94,0\%$ de acurácia na classificação binária no mesmo conjunto de dados.

## 4.4. Atividade Prática (*Hands-On*): Detecção de Ataques em Redes IP Utilizando Redes Neurais Profundas

O objetivo da atividade prática é demonstrar o uso das técnicas de aprendizado profundo discutidas neste minicurso para realizar tarefas de classificação em redes locais. O fluxo de trabalho seguido é o mesmo apresentado na Seção 4.3. Os desempenhos resultantes dos modelos aplicados também são comparados em relação a sistemas baseados em regras. O conjunto de dados escolhido para análise foi o *Intrusion Detection Evaluation Dataset* (CICIDS 2017) [Sharafaldin et al., 2018][1] por ser de fácil acesso e permitir que quase todas as etapas de um processo de análise de dados sejam executadas. Os passos necessários que não são viáveis para execução no contexto da atividade prática, como coleta de dados, são mencionados e discutidos. Uma breve descrição dos dados é feita nesta seção, porém Panigrahi e Borah apresentam uma discussão detalhada sob o ponto de vista da efetividade de sistemas de detecção de intrusão [Panigrahi e Borah, 2018]. Vale mencionar que Panwar et al. realizam uma análise aprofundada sobre o conjunto de dados CICIDS [Panwar e Shailesh, 2019], porém o objetivo desta atividade é permitir que o leitor coloque em prática os conceitos apresentados nas seções anteriores ao invés de buscar uma exploração profunda do conjunto de dados aqui estudado.

[1] Uma descrição detalhada e *links* para *download* podem ser encontrados em `https://www.unb.ca/cic/datasets/ids-2017.html`.

Todos os programas utilizados e descritos nesta seção estão disponíveis em um repositório GitHub[2]. O repositório deve ser clonado para facilitar a execução da atividade prática. Para tal, os seguintes comandos devem ser executados para facilitar a atividade:

- Clonar o repositório:
  `git clone https://github.com/kaylani2/minicurso_ml_sbrc2020`

- Baixar o conjunto de dados:
  `wget http://205.174.165.80/CICDataset/CIC-IDS-2017/Dataset/MachineLearningCSV.zip`

- A partir da raiz do repositório, descomprimir o conjunto de dados:
  `unzip MachineLearningCSV.zip`

- A partir da raiz do repositório, instalar os requisitos:
  `pip3 install -r requirements.txt`

O Algoritmo 1 mostra uma visão de alto nível do treinamento da rede neural para o estudo de caso. Trata-se de um exemplo que considera aprendizado supervisionado.

**Algoritmo 1:** Algoritmo geral de treinamento supervisionado.

**Entrada:** Conjunto de amostras rotuladas.
**Saída:** Matriz de pesos e limiar de ativação.
`/* Inicialização dos conjuntos de dados e da matriz de pesos e limiar de ativação */`
Dividir aleatoriamente o conjunto de dados entre conjuntos de treino, validação e teste.
Dividir o conjunto de treino em lotes (*batches*).
Inicializar aleatoriamente a matriz de pesos e limiar de ativação.
`/* Treinamento do modelo */`
**para cada** *lote em lotes* **faça**
    Aplicar regularização;
    Calcular o valor da função custo no lote atual;
    Calcular o vetor gradiente no lote atual;
    **enquanto** (*custo* $>$ *valor arbitrário*) $\vee$ (*número de épocas* $<$ *épocas escolhidas*) **faça**
        Atualizar os pesos na direção contrária ao vetor gradiente.
    **fim**
**fim**
**retorna** matriz de pesos e limiar de ativação;

### 4.4.1. Apresentação das Bibliotecas de Programação

Inicialmente as bibliotecas e pacotes utilizados na atividade prática são apresentados para familiarizar o leitor com a utilização das estruturas de dados e algoritmos discutidos nas seções anteriores.

[2] `https://github.com/kaylani2/minicurso_ml_sbrc2020`

### *Pandas, Numpy* e *Matplotlib*

O **Pandas**[3] é uma ferramenta de código aberto para análise e manipulação de dados construída sobre a linguagem Python. O Pandas utiliza duas estruturas de dados primárias: `Series` para dados unidimensionais e `Dataframe` para dados bidimensionais, sendo que os `DataFrames` são normalmente utilizados para manipular dados rotulados em sistemas de aprendizado. A biblioteca provê ainda facilidade para manipulação de dados faltantes, métodos comumente utilizados ao trabalhar com múltiplos conjuntos de dados (*join*, *merge* etc.) e ferramentas para converter, agrupar, analisar e modificar o formato dos dados. Normalmente os dados obtidos são carregados na forma de `DataFrames` para uma análise superficial e pré-processamento de dados para serem posteriormente convertidos para outra estrutura de dados apropriada para as bibliotecas que implementam os modelos de aprendizado. O **Numpy**[4] é um pacote largamente utilizado para computação científica. Ele possui as estruturas de dados manipuladas pelas bibliotecas específicas para o aprendizado de máquina e aprendizado profundo. A biblioteca usa vetores (*arrays*) multidimensionais como principal estrutura de dados, conhecida como `ndarray`, comumente tratada pelo seu *alias* `array` ou `numpy.array`. A biblioteca inclui métodos para a manipulação e inspeção dos objetos, além da forma tradicional de acesso a *arrays* através de índices inteiros. O **Matplotlib**[5] é uma biblioteca para exibição de gráficos em múltiplos formatos. Ela pode ser utilizada através de diversas interfaces, porém neste minicurso será manipulada através de programas em Python. O Matplotlib utiliza o elemento `Figure` para exibição de gráficos e figuras através de eixos, no qual a interface `matplotlib.pyplot` é normalmente usada para criação de gráficos simples e interativos. O Matplotlib é uma ferramenta poderosa para análise inicial de conjuntos de dados que facilita a leitura dos resultados de um modelo de aprendizado.

### *Scikit-learn, TensorFlow* e *Keras*

O **Scikit-learn**[6] é uma biblioteca de código aberto para programação de algoritmos de aprendizado de máquina em Python. Ele suporta diversos modelos de aprendizado clássico, porém não implementa modelos de aprendizado profundo. Ela é normalmente utilizada para pré-processamento de dados, além da avaliação de modelos através de métodos para validação cruzada e ajuste de hiperparâmetros. O **TensorFlow**[7] é também uma plataforma de código aberto para implementação de técnicas de aprendizado de máquina. Diferentemente do Scikit-learn, ela é especialmente projetada para aprendizado profundo. A biblioteca utiliza uma estrutura de dados característica chamada `tensor`, que é composta por *arrays* multidimensionais. As funções necessárias para os algoritmos de aprendizado, como o cálculo de produtos matriciais para o *backpropagation*, são otimizadas internamente. O TensorFlow pode ser utilizado genericamente para expressar etapas computacionais arbitrárias através de grafos ou ***flows***, mas é normalmente utilizado para tarefas de aprendizado de máquina e, especificamente, aprendizado profundo. O **Ke-**

[3] `https://pandas.pydata.org/`
[4] `https://numpy.org/`
[5] `https://matplotlib.org/`
[6] `https://scikit-learn.org/`
[7] `https://www.tensorflow.org/`

**ras**[8] é uma biblioteca de código aberto escrita em Python que age como interface para o TensorFlow, sendo uma camada adicional de abstração construída sobre a plataforma TensorFlow. Ao invés de utilizar diretamente a plataforma TensorFlow, é possível selecionar facilmente hiperparâmetros dos modelos profundos, como o número de camadas em uma rede neural, além de diversas funções custo, funções de ativação e otimizadores.

### 4.4.2. Etapas para a Implementação de um Projeto com Aprendizado Profundo

Esta seção descreve de forma resumida as etapas para a implementação de um projeto prático que utilize as técnicas de aprendizado profundo discutidas neste minicurso.

#### Aquisição, Carga e Visualização dos Dados

A primeira etapa para a implementação de um projeto com aprendizado profundo é coletar e armazenar um grande volume de dados, porém a coleta de dados está fora do escopo deste minicurso. Logo, em posse do conjunto de dados CICIDS, especificamente o arquivo `Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv`, é necessário carregar e inspecionar os dados. O arquivo está em formato `CSV` (*Comma-separated values*) e pode ser carregado para inspeção com a biblioteca Pandas. A busca pela visualização gráfica da correlação entre cada um dos atributos e o alvo é uma tarefa impraticável, uma vez que cada amostra contém 78 atributos. Porém, é possível utilizar métodos do pacote Pandas para obter análises estatísticas sobre o conjunto de dados, como o desvio padrão ou a média de um atributo.

#### Tratamento do Conjunto de Dados

É importante notar que o conjunto de dados possui 225.745 amostras rotuladas e menos de $0,01\%$ das amostras possuem informações faltantes. Algumas estratégias podem ser utilizadas para tratar amostras defeituosas como: substituir atributos faltantes pelos valores médios, medianos ou mais frequentes; e eliminar amostras sem rótulos. A fim de apresentar a utilização de bibliotecas para tratamento de dados, um modelo `SimpleImputer` da biblioteca Keras foi utilizada para percorrer o conjunto de dados e substituir os valores de atributos das amostras defeituosas pelo valor médio dos atributos, porquanto substituir esses valores por valores aleatórios pode alterar drasticamente o desempenho do modelo. Visto que alterar os rótulos de amostras desconhecidas *a priori* pode resultar em falsas conclusões sobre os dados examinados e que o número de amostras com atributos faltantes é baixo em relação ao número total de amostras, as entradas "defeituosas" do conjunto de dados também podem ser removidas antes do treinamento. Como o percentual de amostras defeituosas no conjunto de dados é muito pequeno em relação ao total de amostras, ambas as abordagens produzem o mesmo resultado.

#### Configuração da Rede Neural

A biblioteca Keras é utilizada como API para instanciar o modelo profundo. Uma arquitetura com múltiplas camadas ocultas e volume de neurônios progressivamente menor é escolhida. A primeira rede neural escolhida, parcialmente descrita no Código 4.1, é um perceptron de múltiplas camadas (MLP) com uma configuração de camadas densas e

[8] `https://keras.io/`

camadas *dropout*, responsáveis pela regularização durante o treinamento. As camadas de *dropout* escolhem aleatoriamente um número de neurônios em cada camada, de acordo com a probabilidade passada como parâmetro, para ser "desativado" durante uma etapa do treinamento, isto é, sempre produzirá saída nula. Isso tem como objetivo evitar que neurônios da rede sejam sobreutilizados para a tarefa de predição, de forma que o modelo não fique dependente de um conjunto fixo de neurônios.

**Código 4.1. Nova instância de um *Multilayer Perceptron.***

```
################################################################
## Instanciar o modelo de aprendizado
################################################################
from keras.models import Sequential
from keras.layers import Dense, Dropout
numberOfClasses = 2 # Classificacao binaria: Benign, DoS
batchSize = 128
numberOfEpochs = 20
model = Sequential ()
model.add (Dense (512, activation = 'relu', input_shape= (78, )))
model.add (Dropout (0.2))
model.add (Dense (256, activation = 'relu'))
model.add (Dropout (0.2))
model.add (Dense (128, activation = 'relu'))
model.add (Dropout (0.2))
model.add (Dense (numberOfClasses, activation = 'softmax'))
print ('Model summary:')
model.summary ()
```

### Treinamento e Teste do Aprendizado

O Código 4.2 apresenta o trecho necessário para configurar a rede e iniciar o treinamento. Notam-se as escolhas de função de perda, otimizador, taxa de aprendizado e métricas utilizadas durante o treinamento. É importante também notar outros hiperparâmetros utilizados durante o treinamento: tamanho dos conjuntos de treino, número de épocas de treinamento e proporção do conjunto de validação.

**Código 4.2. Compilação e treinamento da Rede Neural.**

```
################################################################
## Compilar a rede
################################################################
from keras.optimizers import Adam
model.compile (loss = 'sparse_categorical_crossentropy',
               optimizer = Adam (lr=0.001),
               metrics= ['accuracy'])

################################################################
## Treinar (fit) a rede
################################################################
history = model.fit (X_train, y_train,
                     batch_size = batchSize,
                     epochs = numberOfEpochs,
                     verbose = 1,
                     validation_data = (X_test, y_test))
```

### Análise dos Resultados

A configuração escolhida, após o ajuste de hiperparâmetros, atinge uma acurácia superior a 90%. Os hiperparâmetros podem ser variados a fim de entender como a variação de arquitetura da rede neural afeta o desempenho do classificador. É importante notar que ajustes no número de neurônios por camada, número de camadas, funções de ativação e otimizadores afetam o desempenho do modelo em conjunto. Dessa forma, não é possível variar o número de neurônios em uma camada, por exemplo, para depois variar o número de camadas em busca do melhor desempenho.

Também é importante mencionar que o algoritmo de otimização escolhido (*Stochastic Gradient Descent* – SGD) possui natureza estocástica, isto é, existem múltiplos caminhos que o algoritmo pode seguir durante a otimização e o resultado pode variar, mesmo que minimamente.

## 4.5. Considerações Finais, Perspectivas e Problemas em Aberto

Este minicurso apresentou conceitos básicos de aprendizado de máquina essenciais para a compreensão do aprendizado profundo. O minicurso ressaltou que o aprendizado profundo é um caso especial de aprendizado de máquina, no qual o modelo de aprendizado é dividido em múltiplas camadas de processamento que se completam de forma hierárquica para resolver um problema complexo. Apesar de existirem outros modelos de aprendizado profundo, o foco deste minicurso foi nas redes neurais profundas, uma vez que essas redes são um componente fundamental desse tipo de aprendizado. Ademais, este minicurso apresentou uma atividade prática na qual foi possível observar os desafios e os benefícios do uso de algoritmos de aprendizado profundo para resolução de problemas complexos. Dentre os principais benefícios, destaca-se a capacidade de tratar um grande volume de dados e a possibilidade de ajustar os hiperparâmetros de um modelo para que ele opere de forma otimizada.

O aprendizado profundo já é amplamente aplicado no processamento de imagens e vem sendo cada vez mais aplicado nas chamadas Redes Desafiadoras. Essas redes são definidas neste minicurso como redes caracterizadas pela geração de um grande volume de dados e pela inerente complexidade de operação. Dentro dessa classificação estão as redes sem fio, IoT, veiculares e industriais. Além de discutir diversas pesquisas que usam o aprendizado profundo para resolver problemas característicos dessas redes, este minicurso também discutiu trabalhos que aplicam aprendizado profundo na resolução de problemas de segurança em redes, que são comuns a todas as Redes Desafiadoras. O minicurso apresentou a forma com que as abordagens utilizam o aprendizado profundo e como ocorre a extração de dados dos conjuntos de dados utilizados, destacando os desafios encontrados nos trabalhos. Um dos desafios comum a todas as redes, por exemplo, é a manutenção de uma base de dados atualizada que preserve a privacidade dos usuários e mantenha as informações importantes anonimizadas, enquanto garante, ao mesmo tempo, que a base de dados contenha todas as informações necessárias para o funcionamento adequado dos algoritmos de aprendizado profundo. Além disso, as informações não podem ser tendenciosas.

As propostas baseadas em redes neurais profundas são capazes de lidar com o crescimento exacerbado do volume de dados a serem analisados e com o aumento da

velocidade de execução dos modelos. A aplicação das diversas arquiteturas baseadas em redes neurais profundas também possibilita que as métricas de avaliação de desempenho, como a acurácia, sejam melhores quando comparadas às obtidas em modelos tradicionais de aprendizado de máquina. Esse melhor desempenho das arquiteturas de aprendizado profundo está relacionado ao ajuste fino dos hiperparâmetros.

Apesar dos resultados promissores alcançados nos trabalhos listados, ainda é necessário aprofundar a pesquisa sobre aprendizado profundo aplicado às Redes Desafiadoras em diversos aspectos. Por exemplo, uma das promessas do aprendizado profundo é permitir a utilização da base de dados sem que haja tratamento prévio. No entanto, isso ainda não é realidade com os algoritmos atuais, uma vez que os dados de entrada das redes neurais profundas ainda são previamente analisados. Além disso, o bom desempenho das soluções baseadas em aprendizado profundo depende do ajuste adequado dos hiperparâmetros para o problema investigado. Esse processo de ajuste é, muitas vezes, automatizado [Aceto et al., 2019a, Wu et al., 2019, Neary, 2018]. No entanto, a maioria dos estudos atuais utilizam métodos empíricos que podem não chegar a soluções ótimas. O uso de valores não ótimos para os hiperparâmetros em geral resulta em resultados ruins quando comparados aos resultados alcançados quando algoritmos de aprendizado clássico são utilizados. Ressalta-se que não existe um modelo único que atenda a todas as questões em aberto, portanto, é necessário ponderar sobre quais modelos devem ser utilizados para atender melhor o problema investigado. Essa investigação é complexa e requer a combinação de diversos tipos de redes neurais profundas para obter bons resultados. Dessa forma, o uso de algoritmos de aprendizado profundo nas Redes Desafiadoras ainda apresenta desafios de pesquisa a serem explorados.

## Agradecimentos

Os autores gostariam primeiramente de agradecer os professores Heraldo Luís Silveira de Almeida e José Gabriel Rodríguez Carneiro Gomes por toda atenção gentilmente dispensada. Em seguida, os autores agradecem igualmente o apoio do CNPq; da Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ); da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES), Código de Financiamento 001; e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processos n$^o$ 15/24494-8 e 15/24490-2.

## Referências

[Abu Alsheikh et al., 2016] Abu Alsheikh et al. (2016). Rate-distortion balanced data compression for wireless sensor networks. *IEEE Sensors Journal*, 16(12):5072–5083.

[Aceto et al., 2019a] Aceto, G., Ciuonzo, D., Montieri, A. e Pescapé, A. (2019a). Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE Transactions on Network and Service Management*, 16(2):445–458.

[Aceto et al., 2019b] Aceto, G., Persico, V. e Pescapé, A. (2019b). A survey on information and communication technologies for industry 4.0: State-of-the-art, taxonomies,

perspectives, and challenges. *IEEE Communications Surveys & Tutorials*, 21(4):3467–3501.

[Al-Hawawreh et al., 2018] Al-Hawawreh, M., Moustafa, N. e Sitnikova, E. (2018). Identification of malicious activities in industrial internet of things based on deep learning models. *Journal of Information Security and Applications*, 41:1–11.

[Alain e Bengio, 2014] Alain, G. e Bengio, Y. (2014). What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593.

[Barbosa et al., 2019] Barbosa et al. (2019). Centralidade de proximidade por múltiplos caminhos disjuntos: Aplicação em redes de longa distância. Em *Anais do SBRC 2019*, volume 37, p. 88–101.

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D. e Larochelle, H. (2007). Greedy layer-wise training of deep networks. Em *Advances in neural information processing systems*, p. 153–160.

[Bengio et al., 1994] Bengio, Y., Simard, P. e Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Bhattacharyya et al., 2019] Bhattacharyya, R., Bura, A., Rengarajan, D., Rumuly, M., Shakkottai, S., Kalathil, D., Mok, R. K. e Dhamdhere, A. (2019). Qflow: A reinforcement learning approach to high qoe video streaming over wireless networks. Em *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, p. 251–260.

[Bianchi et al., 2019] Bianchi, V., Bassoli, M., Lombardo, G., Fornacciari, P., Mordonini, M. e De Munari, I. (2019). IoT wearable sensor and deep learning: An integrated approach for personalized human activity recognition in a smart home environment. *IEEE Internet of Things Journal*, 6(5):8553–8562.

[Charte et al., 2018] Charte et al. (2018). A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines. *Information Fusion*, 44:78–96.

[Comarela et al., 2019] Comarela et al. (2019). Introdução à ciência de dados: Uma visão pragmática utilizando python, aplicações e oportunidades em redes de computadores. Em *Minicursos do SBRC 2019*, chapter 6, p. 246–295. SBC.

[Costa et al., 2012] Costa et al. (2012). Grandes massas de dados na nuvem: Desafios e técnicas para inovação. Em *Minicursos do SBRC 2012*, chapter 1, p. 1–58. SBC.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

[Deisenroth et al., 2019] Deisenroth, M. P., Faisal, A. A. e Ong, C. S. (2019). *Mathematics for machine learning*. Cambridge University Press Cambridge.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y. e Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Grando et al., 2019] Grando et al. (2019). Machine learning in network centrality measures: Tutorial and outlook. *ACM Computing Surveys*, 51(5):102:1–102:32.

[Graves, 2012a] Graves, A. (2012a). Long short-term memory. Em *Supervised sequence labelling with recurrent neural networks*, p. 34–42. Springer.

[Graves, 2012b] Graves, A. (2012b). Neural networks. Em *Supervised sequence labelling with recurrent neural networks*, p. 13–33. Springer.

[Gubbi et al., 2013] Gubbi, J., Buyya, R., Marusic, S. e Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660.

[Hammer, 2000] Hammer, B. (2000). On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1-4):107–123.

[Hammerla et al., 2016] Hammerla, N. Y., Halloran, S. e Plötz, T. (2016). Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv preprint arXiv:1604.08880*.

[Haykin, 1994] Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

[Hinton e Salakhutdinov, 2006] Hinton, G. E. e Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., White, H. et al. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L. e Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285.

[Khan et al., 2018] Khan, S., Rahmani, H., Shah, S. A. A. e Bennamoun, M. (2018). A guide to convolutional neural networks for computer vision. *Synthesis Lectures on Computer Vision*, 8(1):1–207.

[Lara e Labrador, 2012] Lara, O. D. e Labrador, M. A. (2012). A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials*, 15(3):1192–1209.

[LeCun et al., 1995] LeCun, Y., Bengio, Y. et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

[LeCun et al., 2015] LeCun, Y., Bengio, Y. e Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

[Li et al., 2018] Li et al. (2018). Deep Learning for Smart Industry: Efficient Manufacture Inspection System with Fog Computing. *IEEE Transactions on Industrial Informatics*, 14(10):4665–4673.

[Liang e Srikant, 2016] Liang, S. e Srikant, R. (2016). Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*.

[Lv et al., 2015] Lv et al. (2015). Traffic Flow Prediction with Big Data: A Deep Learning Approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873.

[Ma et al., 2019] Ma, X., Yao, T., Hu, M., Dong, Y., Liu, W., Wang, F. e Liu, J. (2019). A survey on deep learning empowered IoT applications. *IEEE Access*, 7:181721–181732.

[Mao et al., 2018] Mao et al. (2018). Deep learning for intelligent wireless networks: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 20(4):2595–2621.

[Medeiros et al., 2016] Medeiros, D. S. V., Campista, M. E. M., Mitton, N., Dias de Amorim, M. e Pujolle, G. (2016). Weighted betweenness for multipath networks. Em *Proc. of the Global Information Infrastructure and Networking Symposium (GIIS '16)*, p. 1–6.

[Medeiros et al., 2017] Medeiros et al. (2017). The power of quasi-shortest paths: $\rho$-geodesic betweenness centrality. *IEEE Transactions on Network Science and Engineering*, 4(3):187–200.

[Medeiros et al., 2019] Medeiros et al. (2019). Análise de dados em redes sem fio de grande porte: Processamento em fluxo em tempo real, tendências e desafios. Em *Minicursos do SBRC 2019*, chapter 4, p. 142–195. SBC.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill.

[Neary, 2018] Neary, P. (2018). Automatic hyperparameter tuning in deep convolutional neural networks using asynchronous reinforcement learning. Em *2018 IEEE International Conference on Cognitive Computing (ICCC)*, p. 73–77.

[Nguyen e Armitage, 2008] Nguyen, T. T. e Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10(4):56–76.

[Osherson et al., 1991] Osherson et al. (1991). A universal inductive inference machine. *Journal of Symbolic Logic*, 56(2):661–672.

[Paine et al., 2014] Paine, T. L., Khorrami, P., Han, W. e Huang, T. S. (2014). An analysis of unsupervised pre-training in light of recent advances. *arXiv preprint arXiv:1412.6597*.

[Panigrahi e Borah, 2018] Panigrahi, R. e Borah, S. (2018). A detailed analysis of cicids2017 dataset for designing intrusion detection systems. *International Journal of Engineering & Technology*, 7:479–482.

[Panwar e Shailesh, 2019] Panwar, Lokesh, P. e Shailesh (2019). Implementation of machine learning algorithms on cicids-2017 dataset for intrusion detection using weka. *International Journal of Recent Technology and Engineering (IJRTE)*, 8:2195–2207.

[Peng et al., 2018] Peng, Z., Gao, S., Li, Z., Xiao, B. e Qian, Y. (2018). Vehicle safety improvement through deep learning and mobile sensing. *IEEE network*, 32(4):28–33.

[Pierucci e Micheli, 2016] Pierucci, L. e Micheli, D. (2016). A neural network for quality of experience estimation in mobile communications. *IEEE MultiMedia*, 23(4):42–49.

[Ravi et al., 2016a] Ravi, D., Wong, C., Lo, B. e Yang, G.-Z. (2016a). A deep learning approach to on-node sensor data analytics for mobile or wearable devices. *IEEE journal of biomedical and health informatics*, 21(1):56–64.

[Ravi et al., 2016b] Ravi, D., Wong, C., Lo, B. e Yang, G.-Z. (2016b). Deep learning for human activity recognition: A resource efficient implementation on low-power devices. Em *2016 IEEE 13th international conference on wearable and implantable body sensor networks (BSN)*, p. 71–76. IEEE.

[Reis et al., 2020] Reis, L. H. A., Magalhães, L. C. S., de Medeiros, D. S. V. e Mattos, D. M. F. (2020). An unsupervised approach to infer quality of service for large-scale wireless networking. *Journal of Network and Systems Management*. In Press.

[Rifai et al., 2011] Rifai, S., Vincent, P., Muller, X., Glorot, X. e Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. Em *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, p. 833–840.

[Sharafaldin et al., 2018] Sharafaldin et al. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. Em *International Conference on Information Systems Security and Privacy (ICISSP)*, p. 108–116.

[Sharma et al., 2019] Sharma, A., Vans, E., Shigemizu, D., Boroevich, K. e Tsunoda, T. (2019). Deepinsight: A methodology to transform a non-image data to an image for convolution neural network architecture. *Scientific Reports*, 9.

[Sisinni et al., 2018] Sisinni, E., Saifullah, A., Han, S., Jennehag, U. e Gidlund, M. (2018). Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, 14(11):4724–4734.

[Srivastava, 2013] Srivastava, N. (2013). Improving neural networks with dropout. *University of Toronto*, 182(566):7.

[Sun e Willmann, 2019] Sun, D. e Willmann, S. (2019). Deep learning-based dependability assessment method for industrial wireless network. *IFAC-PapersOnLine*, 52(24):219–224.

[Tang et al., 2017] Tang, F., Mao, B., Fadlullah, Z. M., Kato, N., Akashi, O., Inoue, T. e Mizutani, K. (2017). On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control. *IEEE Wireless Communications*, 25(1):154–160.

[Vinayakumar et al., 2019] Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A. e Venkatraman, S. (2019). Deep Learning Approach for Intelligent Intrusion Detection System. *IEEE Access*, 7:41525–41550.

[Wang et al., 2016a] Wang, J., Zhang, X., Gao, Q., Yue, H. e Wang, H. (2016a). Device-free wireless localization and activity recognition: A deep learning approach. *IEEE Transactions on Vehicular Technology*, 66(7):6258–6267.

[Wang et al., 2018] Wang, X., Zhou, Z., Xiao, F., Xing, K., Yang, Z., Liu, Y. e Peng, C. (2018). Spatio-temporal analysis and prediction of cellular traffic in metropolis. *IEEE Transactions on Mobile Computing*, 18(9):2190–2202.

[Wang et al., 2017] Wang, Y., Yang, A., Chen, X., Wang, P., Wang, Y. e Yang, H. (2017). A deep learning approach for blind drift calibration of sensor networks. *IEEE Sensors Journal*, 17(13):4158–4171.

[Wang et al., 2016b] Wang, Y., Yang, A., Li, Z., Chen, X., Wang, P. e Yang, H. (2016b). Blind drift calibration of sensor networks using sparse bayesian learning. *IEEE Sensors Journal*, 16(16):6249–6260.

[Wu et al., 2019] Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H. e Deng, S.-H. (2019). Hyperparameter optimization for machine learning models based on bayesian optimizationb. *Journal of Electronic Science and Technology*, 17(1):26 – 40.

[Ye et al., 2019] Ye, H., Li, G. Y. e Juang, B.-H. F. (2019). Deep reinforcement learning based resource allocation for v2v communications. *IEEE Transactions on Vehicular Technology*, 68(4):3163–3173.

[Youssef et al., 2007] Youssef, M., Mah, M. e Agrawala, A. (2007). Challenges: device-free passive localization for wireless environments. Em *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, p. 222–229.

[Yu et al., 2018] Yu, T., Wang, X. e Shami, A. (2018). Uav-enabled spatial data sampling in large-scale IoT systems using denoising autoencoder neural network. *IEEE Internet of Things Journal*, 6(2):1856–1865.

[Zafari et al., 2019] Zafari, F., Gkelias, A. e Leung, K. K. (2019). A survey of indoor localization systems and technologies. *IEEE Communications Surveys & Tutorials*, 21(3):2568–2599.

[Zhang et al., 2018] Zhang et al. (2018). An efficient deep learning model to predict cloud workload for industry informatics. *IEEE Transactions on Industrial Informatics*, 14(7):3170–3178.

[Zhao et al., 2019] Zhao, L., Huang, H., Li, X., Ding, S., Zhao, H. e Han, Z. (2019). An accurate and robust approach of device-free localization with convolutional autoencoder. *IEEE Internet of Things Journal*, 6(3):5825–5840.

[Zhong et al., 2017] Zhong, R. Y., Xu, X., Klotz, E. e Newman, S. T. (2017). Intelligent manufacturing in the context of industry 4.0: a review. *Engineering*, 3(5):616–630.

# Capítulo 5

# Computação Serverless: Conceitos, Aplicações e Desafios

André G. Vieira[1], Gustavo Pantuza[1], Jean H. F. Freire[1], Lucas F. S. Duarte[2], Racyus D. G. Pacífico[1], Marcos A. M. Vieira[1], Luiz F. M. Vieira[1], José A. M. Nacif[2]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
CEP: 31270-901 – Belo Horizonte – MG – Brasil
{andregarcia,pantuza,jean,racyus,mmvieira,lfvieira}@dcc.ufmg.br

[2]Instituto de Ciências Exatas e Tecnológicas
Universidade Federal de Viçosa (UFV)
CEP: 35690-000 – Florestal – MG – Brasil
{lucas.f.duarte, jnacif}@ufv.br

***Abstract***

*Serverless computing is a novel application deployment model that provides scaling, fine-grained billing, and platform decoupling for developers. This model has the potential for changing the landscape of technology development for users and cloud providers. In this short course, we present the Serverless Computing state of the art: concepts, limitations, use cases, research opportunities, caveats, and a view on some current platforms.*

***Resumo***

*Computação Serverless é um novo modelo de disponibilização de aplicações que provê escalabilidade, precisão de cobrança e desacoplamento de plataforma para desenvolvedores. Este modelo possui potencial para mudar o cenário de desenvolvimento de tecnologias tanto para usuários quanto para provedores de nuvem. Neste minicurso apresentamos o estado da arte em Computação Serverless: seus conceitos, limitações, casos de uso, oportunidades de pesquisa, ressalvas e uma visão sobre algumas plataformas atuais.*

## 5.1. Introdução

Segundo Castro et al. [Castro et al. 2019], é esperado em 2020 que 67% dos gastos com *software* e infraestrutura pelas corporações sejam baseadas em tecnologias em nuvem. Eles também afirmam que essas tecnologias tinham como promessas o modelo de pagar apenas pelo que for utilizado e de abstrair os detalhes de como os servidores eram utilizados e mantidos. A computação *Serverless* surge como um novo modelo de tecnologia em nuvem que permite escalabilidade, precisão de cobrança e desacoplamento de plataforma para desenvolvedores.

### 5.1.1. Histórico

Historicamente o interesse no uso de plataformas virtualizadas cresceu ao longo da popularização da internet e aplicações web. O uso de máquinas virtuais permitiu que um mesmo *datacenter* pudesse ser compartilhado entre diversas aplicações ou mesmo clientes, devido à possibilidade de reaproveitamento de *hardware* e isolamento entre sistemas operacionais. Serviços de hospedagem de máquinas virtuais como AWS EC2 (2006), Google Cloud Compute Engine (2012) ou Azure Virtual Machines (2010) surgiram como negócios neste paradigma. Estas plataformas são comumente encaixadas sob o termo guarda-chuva IaaS *Infrastructure as a Service* – Infraestrutura como um Serviço).

Uma abordagem diferente é utilizada por serviços PaaS (*Platform as a Service* – Plataforma como um Serviço). Neste modelo, não se especifica a infraestrutura na qual o serviço é executado, mas sim a plataforma (bibliotecas, ambientes de execução) na qual se executa. Neste modelo se destacam Heroku (2006) e Google App Engine (2008). Estas plataformas permitem um nível de abstração acima de máquinas virtuais, mas ainda exigem configurações específicas e carecem de padrões de código aberto que facilitem migração de uma plataforma para outra.

A popularização do uso de contêiners levou ao surgimento de ofertas comerciais que os suportassem. Largamente baseados no sucesso do Docker (2013) como principal ferramenta de conteinerização, desenvolvedores e administradores de sistema se tornaram aptos a poder implantar soluções de maneira reproduzível em diferentes fornecedores ou localmente, se necessário - com uma necessidade de configuração automatizável e menor do que um Sistema Operacional completo.

O AWS Lambda (2014) é considerado como o primeiro serviço *Serverless* assim nomeado. Nele, o responsável pela implantação do serviço desenvolve uma função em alguma das linguagens de programação aceitas e esta fica disponível numa URL, sendo ativada de acordo com critérios pré-especificados, tais como uma chamada HTTP ou um evento disparado por uma fila de mensagens.

### 5.1.2. Computação em nuvem moderna

Nos últimos 10 anos as tecnologias em nuvem conseguiram maximizar a utilização por meio de virtualização e reduzir o custo de escalabilidade para grandes *datacenters*. No entanto, os usuários da nuvem continuam a suportar uma carga de operações complexas e nem sempre se beneficiam do compartilhamento de recursos com outros usuários como discutido por Jonas [Jonas et al. 2019].

A computação *Serverless* é a plataforma que abstrai dos desenvolvedores o servidor e roda código sob-demanda que é escalado e contabilizado automaticamente somente pelo tempo em que é executado, assim definido por Castro [Castro et al. 2019]. Além disso, Jonas [Jonas et al. 2019] pontuam que a camada *Serverless* situa-se entre as camadas de aplicação e da plataforma de nuvem subjacentes; simplificando a programação em nuvem.

Em função de sua simplicidade e vantagens econômicas, a computação *Serverless* vem ganhando popularidade como reportado pela taxa de crescimento do termo de busca *Serverless* no Google Trends. O tamanho do mercado é estimado em 7.72 bilhões de dólares até 2021, afirmado por Castro [Castro et al. 2019].

A fundação *Linux Foundation* possui um grupo de trabalho chamado *Cloud Native Computing Foundation* (CNCF), destinada exclusivamente às tecnologias nativamente desenhadas para computação em nuvem. Esse grupo de trabalho possui uma seção dedicada às tecnologias *Serverless* e disponibilizam uma fotografia atualizada com as principais ferramentas em uso desse mercado [CNCF landscape 2020]. Essa fotografia é chamada de *CNCF Serverless Landscape*.

### 5.1.3. Usos da Computação Serverless

Shafiei [Shafiei et al. 2020] afirmam que a computação *Serverless* se difere da computação em nuvem tradicional no sentido em que a infraestrutura e a plataforma às quais os serviços estão executando são de fato transparentes para o usuário da nuvem. Nessa abordagem o desenvolvedor se preocupa exclusivamente com as funcionalidades demandadas por suas aplicações e todo o resto é delegado ao provedor de serviço em nuvem.

As plataformas *Serverless* permitem uma variedade de usos possíveis. Dentre eles, se destacam:

- Invocação periódica de tarefas de rotina, tais como atualizações de bases de dados;
- Responder a eventos disparados numa fila de tarefas;
- Colaboração em tempo real como chats e sistemas de notificação;
- Ferramentas analíticas que recebem grandes volumes de mensagens para processamento individual que podem escalar facilmente em função do volume de mensagens;
- Uso e cobrança sob demanda em aplicações de serviços urbanos planejados para cidades inteligentes;
- Aprendizado de máquina no uso massivo e paralelo das funções;
- Segurança na forma de funções reativas a detecções de intrusão;
- Internet das coisas.

Essas formas de utilização aproveitam das características de tais plataformas. Tarefas de rotina agendadas precisam ser executadas apenas nos momentos necessários, de

modo que pode ser desnecessário manter um servidor executando apenas em espera deste momento. Uma maneira tradicional de se lidar com agendamentos periódicos em sistemas UNIX, por exemplo, é o utilitário *cron*, que depende da execução contínua do sistema para que a tarefa seja executada nos momentos especificados. Com uma plataforma *Serverless*, fica a cargo do provedor do serviço a ativação periódica, enquanto que a função implantada deve apenas ser executada nestes momentos, sendo cobrada apenas por cada ativação e consequente uso de recursos computacionais.

De maneira similar, funções que respondam a um evento disparado numa fila de tarefas apenas serão ativadas quando houverem eventos para serem processados. Outra característica das plataformas *Serverless* que pode ser aproveitada neste caso é a rápida escalabilidade: em momentos nos quais houver muitos eventos para serem processados mais funções podem ser invocadas mais rápido e a um custo tanto financeiro quanto computacional potencialmente menor do que ativar mais máquinas virtuais ou contêineres.

### 5.1.4. Potenciais da computação *Serverless*

Por se tratar de uma tecnologia recente em relação a máquinas virtuais e contêineres há potencial para ser explorado com a compreensão e desenvolvimento na tecnologia. Sob a perspectiva de trabalhos de pesquisa, Hendrickson et al. [Hendrickson et al. 2016] falam sobre depuração orientada a custo financeiro, decomposição facilitada de sistemas legados e compartilhamento de ambientes de execução.

Além disso, a computação *Serverless* é uma camada de serviço que realmente abstrai a infraestrutura de nuvem subjacente, minimizando a curva de aprendizado para que desenvolvedores e projetos utilizem programação em nuvem. Potencialmente, por conseguir contabilizar o uso de forma mais granular, reduz os custos com infraestrutura de nuvem para corporações além de discriminar melhor onde os custos foram alocados.

### 5.1.5. O minicurso

Este minicurso destina-se a apresentar o estado da arte nesta área de conhecimento: Computação *Serverless* (*Serverless Computing* – Computação sem Servidor). Neste modelo de computação, a unidade básica e objeto de trabalho é uma única função escrita numa linguagem de programação de alto nível. Nenhuma das camadas como infraestrutura, máquina virtual, sistema operacional, processo ou *contêiner* são de responsabilidade do usuário da plataforma de computação *Serverless*. Esse modelo requer uma decomposição maior na escrita de aplicações e total dependência sobre a rede e serviços externos. Entretanto, esse modelo fornece capacidades consideráveis de escala conforme discutido por Hendrickson [Hendrickson et al. 2016], granularidade de cobrança e desacoplamento da infraestrutura.

O minicurso está organizado da seguinte forma: A seção 5.2 detalha o paradigma de computação "Função como Serviço". A seção 5.3 explica e detalha o modelo de computação *Serverless*. Na seção 5.4 são apresentadas as principais plataformas. A seção 5.5 mostra projetos de pesquisas utilizando *Serverless*, enquanto na seção 5.6 são mostrados os desafios e limitações em utilizar tal modelo. A seção 5.7 introduz conceitos básicos sobra a plataforma OpenFaas, apresenta também um tutorial de instalação e de como escrever funções, além de descrever alguns exemplos. Finalmente, a seção 5.8 traz a

conclusão e as discussões finais deste minicurso.

## 5.2. Funções como Serviço

Nesta seção se apresenta o paradigma de funções como serviço a partir do exame do conceito "como serviço" em si e em comparação a modelos precursores.

### 5.2.1. Paradigma "como serviço"

O paradigma de disponibilização de aplicações denominado "como serviço" tem essa terminologia devido à ocultação de aspectos do usuário [Wang et al. 2018]. Exemplos do paradigma são:

- Plataforma como serviço (PaaS – *Platform as a Service*). Estas ofertas de computação na nuvem fornecem uma plataforma de execução para aplicações, ocultando do desenvolvedor a infraestrutura subjacente. Exemplos de plataforma como serviço incluem o Google App Engine e o Heroku.

- Infraestrutura como serviço (IaaS – *Infrastructure as a Service*). Esta modalidade consiste no fornecimento de componentes de infraestrutura tais como máquinas virtuais ou redes privadas virtuais (VPS – *Virtual Private Server*). Google Cloud Compute Engine, Amazon Elastic Compute Cloud e Azure VMs são exemplos de infraestrutura como serviço.

- Software como serviço (SaaS – *Software as a Service*). Este modelo compreende o software em si sendo oferecido para um cliente por uma interface (web, por exemplo) mas executando remotamente em servidores do provedor. Exemplos de tal tipo de software são o cliente de e-mail Gmail ou o Microsoft Office Online.

Nesta tendência, o paradigma Funções como serviço (FaaS – *Functions as a Service*) fornece uma perspectiva similar. Ao invés de se provisionar infraestrutura ou a própria plataforma na qual o código é executado, se fornece uma estrutura que irá executar uma unidade de código em si. Abstraído para o desenvolvedor se encontram a plataforma subjacente, sendo necessário apenas fornecer uma listagem de dependências do código; a maneira exata pela qual as execuções de função devem escalar, sendo relacionadas às limitações de escala; e a parte de conectividade da função, sendo que o provedor fornece um identificador de recurso uniforme (*Uniform Resource Identifier* – URI) para a função e o desenvolvedor apenas lida com os dados da requisição e o que deve ser devolvido como resposta.

Entretanto, os detalhes deixados fora do controle do desenvolvedor da aplicação continuam sendo importantes. Uma organização que se utilize de funções como serviço pode integrá-las a uma estrutura maior em seu provedor de computação na nuvem. Um exemplo se encontra nas redes privadas virtuais, para as quais pode-se criar regras sobre quais clientes podem acessar quais funções. O fato de se remover certas preocupações sobre em qual estrutura serão executadas as funções não significa que elas devam ser removidas de toda a cadeia de desenvolvimento.

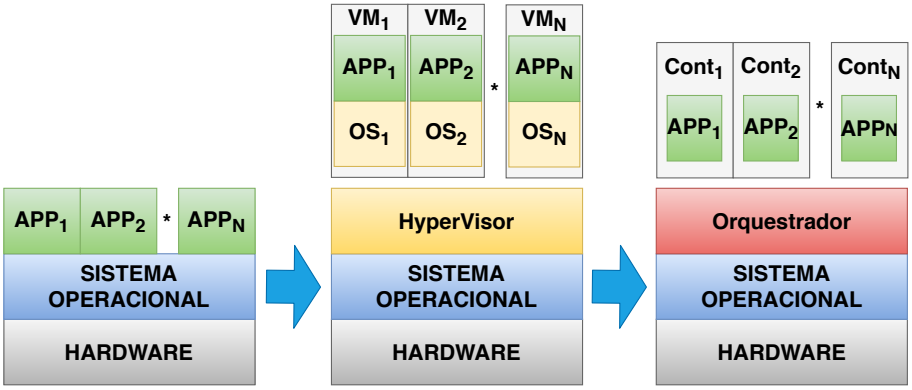



**Figura 5.1. Evolução dos serviços de virtualização.**

### 5.2.2. Evolução: *bare-metal*, virtualização, contêineres

Jonas et al. [Jonas et al. 2019] e Hendrickson et al. [Hendrickson et al. 2016] descrevem um caminho evolutivo na maneira em que aplicações e serviços foram disponibilizados.

Inicialmente, no modelo *bare-metal*, cada aplicação roda diretamente sob uma maquina física, contando apenas com o sistema operacional como intermediário. Apesar das vantagens apresentadas no desempenho da aplicação, visto que todos os recursos da máquina estavam disponíveis para ela, o alto custo e a dificuldade de prover e gerenciar esse tipo de sistema levaram ao advento da virtualização.

Em sistemas virtualizados é adicionada uma camada extra sobre o sistema operacional, chamada de *hypervisor* que permite executar vários sistemas operacionais de maneira isolada sobre o mesmo hardware. Isto resulta em maior flexibilidade e torna a administração destes sistemas mais simples. O gerente da aplicação não precisa mais se preocupar com questões relativas à máquina física e os custos de implantação são reduzidos pois não é mais necessário adquirir um novo servidor para cada aplicação bastando apenas instanciar uma nova máquina virtual.

Mais recentemente surgiram os contêineres, que nada mais são do que um sistema de virtualização em nível de sistema operacional, ou seja, permitem que aplicações sejam executadas de forma isoladas em espaço de usuário. Nestes sistemas as funções básicas do sistema operacional são compartilhadas por todos os contêineres permitindo encapsular as aplicações de modo muito mais enxuto e eficiente que as máquinas virtuais.

A figura 5.1 apresenta um diagrama dessa evolução dos serviços de virtualização. A primeira parte da figura ilustra as aplicações que executam diretamente em cima do sistema operacional. A seguir se observa o sistema virtualizado com a presença do *hypervisor* e as aplicações sendo executadas em máquinas virtuais. Finalmente, na terceira parte da figura pode-se observar as aplicações sendo executadas em contêineres com a presença do orquestrador.

### 5.2.3. Comunicação em contêineres

O modelo de redes para contêineres [Church et al. 2020] fornece uma série de *drivers* de redes nativos que podem ser escolhidos com base nos requisitos da aplicação, são eles

*Host*, *Bridge*, *Overlay*, *MACVLAN* e *None*. Contêineres que utilizam o driver *Host* usam a pilha de rede do hospedeiro sendo que não existe separação de *namespace* e todas as interfaces do hospedeiro podem ser usadas diretamente pelo contêiner.

O *driver Bridge* provê uma rede *bridge* entre o hospedeiro e o contêiner a ser gerenciada pelo orquestrador. Os contêineres conectados à mesma rede *bridge* podem se comunicar entre si provendo isolamento dos que não estão conectados. É o tipo padrão de rede, que o orquestrador usa quando o *driver* não é especificado.

O *driver Overlay* possibilita a comunicação entre mais de um *host* que rodam um orquestrador de contêineres, possibilitando assim a criação de *clusters*. Assim, como em redes do tipo *bridge*, os contêineres conectados a mesma rede *overlay* podem se comunicar, porém neste tipo de rede isso é possível para contêineres rodando em diferentes hospedeiros.

No *driver* do tipo *MACVLAN* cada contêiner tem um endereço MAC atribuído permitindo que ele seja usado com aplicações legadas que precisam de acesso direto à rede local e aplicações que realizam monitoramento de rede. Neste tipo rede é necessário associar cada interface de rede criada a uma interface "pai", geralmente a interface física do hospedeiro, como o acesso é feito diretamente a rede física é necessário um *gateway* para acesso externo, isto permite também o isolamento do trafego através da associação de cada sub-interface criada a uma *Vlan* diferente.

Por fim, o *driver* do tipo *None* é usado para contêineres isolados que não precisam se comunicar nem com outros contêineres nem com redes externas.

## 5.3. Modelo Serverless

O modelo de computação *Serverless* ("sem servidor") é assim chamado não por dispensar por completo o uso de servidores (máquinas, físicas ou virtuais, que disponibilizam aplicações *online*) mas por retirar do desenvolvedor a responsabilidade de configurar o recurso computacional, deixando apenas a escrita das funções que compõem o artefato desenvolvido [Hendrickson et al. 2016].

É possível argumentar que, configuraria um uso do modelo *Serverless*, o caso de aplicações nas quais um grupo de pessoas configura os servidores e outro grupo desenvolve os programas. Desta maneira, entretanto, tem-se uma divisão na qual ainda se tem a necessidade e o conhecimento sobre a plataforma computacional na qual são executados os programas por algum componente do projeto; o modelo *Serverless* deixa a cargo do provedor de serviço da aplicação (por exemplo, AWS [AWS Lambda 2014], Azure [Azure Functions 2016] ou Google Cloud [Google Cloud Functions 2016]) a infraestrutura computacional – desde a disponibilização básica até a escalabilidade do serviço. Nenhum membro da equipe possuiu acesso a uma máquina virtual ou contêiner individual, apenas à aplicação em si.

### 5.3.1. Sem servidor mesmo?

Conforme mencionado na seção 5.3, membros da equipe de desenvolvimento não possuem acesso aos elementos da infraestrutura, sejam eles máquinas físicas, virtuais ou contêineres. Comercialmente se trata de um argumento de vendas que reduz a necessidade

de se gastar recursos com a administração de servidores: a equipe de desenvolvimento consegue colocar em produção um sistema sem interagir com a infraestrutura.

Essa abordagem, no entanto, possui limitações. Funções no modelo *Serverless* são inerentemente sem estado, seguindo o princípio de que elas são horizontalmente escaláveis e independem da infraestrutura subjacente. Assim sendo, sistemas que necessitem de dados persistentes devem acessá-los de outra fonte - bancos de dados, por exemplo. O banco de dados em si deve ser configurado. Fornecedores de serviço na nuvem, como os já citados, fornecem também armazenamento-como-serviço, de maneira que o desenvolvedor pode ligar sua aplicação a algum destes serviços. Esta estrutura limita de fato o contato com a infraestrutura tradicional, mas de certa forma apenas desloca a responsabilidade de um administrador de sistemas convencional para um administrador de sistemas na nuvem. Há vantagens nesta percepção: as tarefas de gerenciamento das máquinas, manutenção e segurança são deslocadas para a plataforma, que em tese possui mais recursos e pessoal para lidar com a infraestrutura do que uma empresa menor. Desvantagens incluem o preço de tais comodidades e o risco de se ficar preso a um fornecedor (*vendor lock-in*).

Com relação aos recursos computacionais, serviços como AWS Lambda ou ferramentas como OpenFaaS [Ellis 2017g] ou OpenLambda [Hendrickson et al. 2016] se utilizam de contêineres para prover isolamento entre as funções e modelo rápido de escala – quanto mais instâncias de execução da função se desejarem simultaneamente, mais contêineres devem ser criados. Estes contêineres, internamente, devem ser controlados via orquestradores como Docker Swarm ou Kubernetes, e toda essa estrutura deve ser executada sobre máquinas tradicionais - virtuais ou físicas. Mais uma vez, todos estes componentes são ocultos do desenvolvedor da aplicação. Assim sendo, no modelo *Serverless* não há servidores visíveis para o usuário final.

### 5.3.2. Responsabilidades da plataforma

Uma plataforma *Serverless* fornece ao usuário uma localidade na qual ele disponibiliza o serviço para utilização, geralmente por meio de uma URL. Esta disponibilidade pode ser combinada e assegurada por meio de um SLA (*Service Level Agreement*). Escalabilidade do serviço, sob os parâmetros definidos pelo cliente, também deve ser fornecida. Maneiras de se inserir, modificar e monitorar o uso da função devem ser providos. Segurança básica contra ataques DDoS ou vírus na plataforma subjacente são considerações importantes.

### 5.3.3. Responsabilidades do desenvolvedor

O desenvolvedor da aplicação deve estar atento ao ecossistema que sua aplicação necessita: bancos de dados, acessos a outras funções e recursos. Conhecimento da plataforma *Serverless* utilizada também é importante: tempo máximo no qual uma função pode ser executada, limites de memória e custo dos recursos utilizados. Uma aplicação pouco otimizada possui um custo diretamente mensurável, então além de se identificar possíveis pontos de melhora há também o monitoramento, utilizando ferramentas da plataforma ou auxiliares. Preocupações com segurança também existem do lado do desenvolvedor: informações expostas, níveis de acesso e configuração de permissões.

### 5.3.4. Precificação

Funções são cobradas por invocação, ou seja, sem custos por recursos não utilizados ou ociosos [Wang et al. 2018]. Nesse modelo, além de fornecer um nível de granularidade que permite medições mais acuradas, o preço também inclui as capacidades de administração de sistemas como redundância, disponibilidade, monitoramento, entre outros [Jonas et al. 2019].

### 5.3.5. Segurança

Mesmo com a mudança do foco da administração para o provedor da plataforma *Serverless* o desenvolvedor ainda possui responsabilidades com a segurança da aplicação. A mudança de foco em si não exime que se tome este cuidado; de fato, apenas modifica como deve ser realizado ou mesmo insere novos desafios. Em [Jonas et al. 2019] os autores citam como exemplos:

- Randomização de escalonamento e isolamento físico;
- Contextos de segurança de fina granularidade;
- Computação *Serverless* "esquecível".

### 5.3.6. Modelo Serverful

*Serverful* é o nome usado por Jonas et al. [Jonas et al. 2019] para descrever a abordagem tradicional dos sistemas em nuvem. Nesta abordagem, o desenvolvedor aluga a infraestrutura de Tecnologia da Informação (TI), como servidores, máquinas virtuais, armazenamento, redes e sistemas operacionais, de um provedor de nuvem, e executa e disponibiliza suas aplicações através dessa infraestrutura.

Porém, diferente do modelo descrito na seção 5.3, ainda é de responsabilidade do desenvolvedor a configuração dos pacotes e serviços necessários à aplicação. O desenvolvedor tem acesso direto a máquina virtual ou contêiner e é o responsável por gerí-lo. No que tange a precificação, o pagamento é realizado por *slots* de tempo, mesmo quando os recursos não estão sendo utilizados. Em casos em que a demanda aumenta, uma nova máquina virtual e/ou contêiner deve ser instanciado, aumentado os custos e, potencialmente, desperdiçando recursos.

Emfim, apesar do modelo *Serverful* ter retirado do desenvolvedor as preocupações com a compra e gerência de hardware, ele ainda não conseguiu abstrair completamente as questões de infraestrutura, objetivo melhor alcançado pelo modelo *Serverless*.

### 5.3.7. Modelo Serverful x Modelo Serverless

Segundo Jonas et al. [Jonas et al. 2019], o ganho do modelo *Serverless* em relação ao modelo *Serverful* pode ser comparado a transição entre linguagens de alto e baixo nível. Enquanto as linguagens de alto nível libertaram o desenvolvedor do gerenciamento de memória, o modelo *Serverless* libertou-o da gerência do servidor de aplicação.

Existem três principais diferenças do modelo *Serveless* em relação ao *Serverful*, são elas:

**Tabela 5.1. Comparação entre os modelos *Serverful* e *Serveless*.**

| | *Serverful* | *Serverless* |
|---|---|---|
| **Escalabilidade** | É necessário subir novas máquinas e/ou contêineres caso a demanda aumente. Em caso de baixa demanda, recursos são desperdiçados. | Escala automaticamente, não é necessário nenhuma ação por parte do desenvolvedor da aplicação |
| **Manutenção** | Requer manutenção, é preciso instalar, monitorar e manter atualizados os softwares necessários a aplicação | Não requer manutenção, isso fica a cargo do provedor do serviço. Basta ao desenvolvedor escrever e implantar código usando as ferramentas fornecidas por esse provedor. |
| **Custo** | É pago por tempo para manter o servidor disponível mesmo que não esteja sendo usado. | É pago por invocação, caso não ocorra o uso não existe cobrança. |
| **Implantação** | Permite implantar serviços diretamente. É necessário logar na maquina e rodar o serviço e suas dependências. | É baseado em eventos, o desenvolvedor implanta uma função e ela é executada com resposta a um evento que ocorre no sistema (*triggers* de banco de dados, requisições HTTP). |

- O armazenamento e a computação ocorrem em separado, geralmente são ofertados por diferentes serviços de nuvem e a computação é sem estado.
- O desenvolvedor fornece apenas um pedaço de código a ser executado. Os recursos para executá-lo são totalmente fornecidos pelo provedor do serviço.
- O pagamento é realizado pelo recurso usado e não pelo recurso alocado.

A tabela 5.1 apresenta a sumarização das principais características de ambos.

## 5.4. Plataformas

Atualmente existem diversas plataformas de computação *Serverless* disponíveis. As plataformas podem ser pagas ou de código livre. Esta seção apresenta algumas dessas plataformas, e discute alguns aspectos sobre cada uma dessas das plataformas. É apresentada uma descrição mais detalhada da plataforma OpenFaaS, devido à escolha dela para a parte prática desse minicurso.

### 5.4.1. AWS Lambda

A AWS Lambda [AWS Lambda 2014] é a plataforma sem servidor da Amazon e faz parte do pacote de serviços Amazon Web Services (AWS). O serviço de computação em nuvem da Amazon foi lançado em 2006 e a AWS Lambda foi lançado em 2014. Segundo

pesquisa realizada pela FLEXERA [Flexera 2019] a AWS é a plataforma em nuvem mais utilizada pelas empresas consultadas. Em consequência desse domínio do mercado, a plataforma de computação *Serverless* AWS Lambda também domina parte significativa do mercado de funções como serviço. A AWS Lambda foi responsável pela popularização do serviço e por moldar, em partes, o ecossistema de aplicações sem servidor.

A AWS Lambda é, como o modelo de computação *Serverless*, orientada a eventos, o que significa dizer que as funções Lambdas são executas como resposta a algum evento. Os eventos podem ser desde uma requisição HTTP, alguma solicitação de alguma aplicação específica ou algum gatilho periódico contido nas configurações. A AWS não fornece detalhes sobre a arquitetura ou funcionamento da AWS Lambda. Contudo, as funções usam um serviço personalizado, chamado Firecracker, para criar máquinas virtuais leves.

A AWS já conta com um extenso escopo de serviços e plataformas de computação em nuvem. Sendo assim, uma das vantagens em utilizar sua plataforma de computação sem servidor reside no fato de ser facilmente integrado com as outras aplicações e serviços oferecidos. Além disso, o AWS Lambda conta com vasta documentação, tanto da própria Amazon quanto de usuários de seus serviços.

Apesar de a AWS Lambda ser um serviço pago, a Amazon oferece 12 meses de acesso gratuito. O acesso ao nível gratuito para os serviços AWS, até o momento da escrita desse minicurso, fornece até 1 milhão de chamada de funções por mês. A AWS fornece uma granularidade de 100 milissegundos para o tempo de execução das funções, o que resultaria em 100 mil segundos de computação. Entretanto, o nível gratuito oferece mais de 3 milhões de segundos de computação.

As funções *Serverless* executadas na plataforma AWS Lambda podem ser escritas utilizando diversas linguagens. Até o momento da escrita desse minicurso as linguagens suportadas pela AWS Lambda são, Java, Go, PowerShell, JavaScript[1], C#, Python e Ruby.

### 5.4.2. Azure Functions

A Azure Functions [Azure Functions 2016] é a plataforma sem servidor da Azure. A Azure é a oferta de computação em nuvem oferecida pela Microsoft, sendo lançada em 2010 inicialmente sob nome de Windows Azure, sendo renomeada para Microsoft Azure. Segundo a pesquisa da FLEXERA [Flexera 2019] a Microsoft Azure é a segunda oferta *cloud* mais utilizada no mercado, ficando atrás somente da AWS.

Assim como a AWS, a Microsoft Azure não disponibiliza maiores detalhes sobre a arquitetura da Azure Functions. Entretanto, o Azure Functions oferece um subproduto chamado *Durable Functions* cujo serviço permite escrever funções com estado dentro de um ambiente *Serveless*. Contudo, esse serviço só consegue manter estado entre duas chamadas de funções do modelo sem servidor, funcionando mais como um *checkpoint* entre duas funções e possui certas restrições como de linguagem ou de serviços suportados.

A Microsoft Azure conta com vários serviços de computação em nuvem e podem ser facilmente integrados com o Azure Functions. Além disso pode ser usado com segurança em requisições HTTP de outros provedores como Facebook, Google e Twitter. No momento de escrita deste artigo, o Azure Functions suporta as linguagens C#, JavaS-

[1] Através do interpretador Node.js

cript[2], F#, Java, PowerShell, Python e TypeScript. Já o ambiente Durable Functions dá suporte somente as linguagens C#, JavaScript e F#.

O serviço Microsoft Azure também é um serviço pago. Entretanto, também oferece 12 meses gratuitos. Ele conta com vasta documentação da Microsoft.

### 5.4.3. Cloud Functions

A Cloud Functions [Google Cloud Functions 2016] é a plataforma *Serverless* da Google Cloud. A Google começou a oferecer serviços de computação em nuvem em 2008, passando a integrar computação sem servidor em 2016. O Cloud Functions promete execução local ou em nuvem. Assim como descrito nas seções 5.4.1 e 5.4.2 a Cloud Functions não disponibiliza maiores detalhes de sua arquitetura.

Assim como as demais plataformas a Cloud Functions não necessita de provisionamento de recursos e se integra facilmente aos demais serviços oferecidos pela plataforma Cloud. A Google também oferece 12 meses de acesso gratuito a plataforma.

A Cloud Functions, dentre as 3 plataformas citadas até aqui, é a que dispõe de menos documentação sobre seus serviços. As linguagens suportadas até o momento da escrita desse minicurso são JavaScript, Go e Python.

### 5.4.4. OpenFaaS

Começou como um projeto pessoal de Alex Ellis em Outubro de 2016. Alex queria executar funcionalidades da Alexa e funções Lambdas como serviços sobre Docker Swarm. Em Dezembro de 2016, com o sucesso inicial de seus resultados, Alex lançou a primeira versão que denominou OpenFaaS, escrita na linguagem Golang e publicada no GitHub [Ellis 2016]. Após a primeira publicação, o projeto ganhou mais de 4.000 estrelas no GitHub impulsionando o conhecimento do projeto na comunidade e indústria da área. Em Abril de 2017, em Austin, Alex ganhou a oportunidade de participar da sessão de palestras da Moby's Cool Hacks em Dockercon. A missão de Alex era usar o OpenFaaS para ultrapassar os limites de desempenho do Docker, onde obteve sucesso. Após estes acontecimentos o projeto do OpenFaaS tornou-se uma plataforma *Serverless* com suporte de 136 colaboradores ativos em seu repositório oficial no GitHub [Ellis 2017f].

A plataforma OpenFaaS é de código aberto e baseado na licença MIT [MIT 1980]. Atualmente, OpenFaaS presta serviços comerciais, suporte a usuários, e gerenciamento de patrocínios na página oficial através do OpenFaaS Ltd responsável pela marca. No OpenFaaS, funções são executadas sobre contêineres de forma escalável baseado no aumento do número de funções. Além disso, todo o gerenciamento de operações com funções pode ser realizada via interface gráfica ou interface de comando atendendo níveis de usuários diferentes, do iniciante ao avançado. A plataforma do OpenFaaS também pode rodar sobre hardwares existentes no mercado, ou em servidores de nuvem privados ou públicos sendo totalmente portátil entre ambos ambientes.

Funções *Serverless* sobre OpenFaaS podem ser escritas nas linguagens de programação Go, C#, JavaScript, Java, Ruby e PHP. Estas linguagens são oficialmente suportadas pela plataforma, no entanto, usuários podem adicionar outras linguagens. Na

[2]Através do interpretador Node.js

seção 5.7.3 descrevemos como adicionar uma nova linguagem na plataforma.

#### 5.4.4.1. Arquitetura

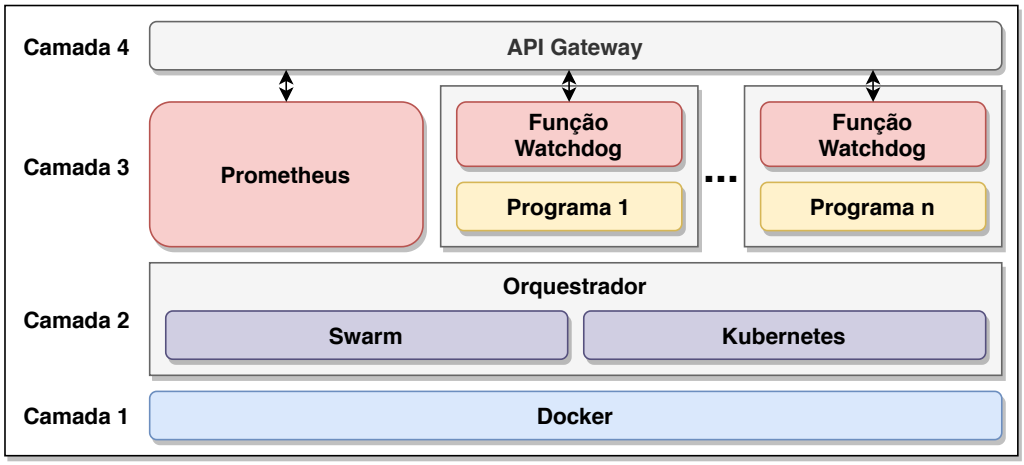



**Figura 5.2. Pilha da arquitetura OpenFaaS.**

A arquitetura do OpenFaaS é baseada no modelo de pilha de componentes. Neste modelo qualquer processo pode ser transformado em função *Serverless* sobre um contêiner Docker. A figura 5.2 apresenta a pilha da arquitetura OpenFaaS composta por quatro camadas. Nesta seção descrevemos cada camada da pilha OpenFaaS na ordem *top-down*, ou seja, da camada quatro (mais abstrata) até a camada um (menos abstrata).

**Camada 4:** Essa camada tem como objetivo gerenciar o processo de comunicação entre usuários e funções *Serverless* como serviços utilizando o protocolo HTTP. O componente API Gateway é o responsável por gerenciar esse processo. Ele recebe e redireciona requisições HTTP enviadas entre usuários e serviços de acordo com a operação e status do serviço. Além disso, ele cria e exclui réplicas de serviços usando o orquestrador (Camada 2) de acordo com o aumento no número de requisições. O API Gateway ainda utiliza uma interface Web intuitiva, denominada Portal UI para facilitar a iteração do usuário com serviços no OpenFaaS. Como medida de segurança e desempenho o API Gateway aloca um contêiner próprio sobre OpenFaaS, evitando que uma função seja requisitada mais de uma vez enquanto estiver sendo utilizada por um usuário específico.

**Camada 3**: Ocorre o processo de comunicação entre o usuário e funções *Serverless*, e o monitoramento de estatísticas do OpenFaaS. Uma função criada pelo usuário será executada em um contêiner específico. Por padrão, o contêiner da função é composto por uma instância da função chamado *Watchdog*, e um programa com a função a ser processada no contêiner. O programa com a função pode ser escrito em qualquer linguagem de programação desde que adicionado o suporte no OpenFaaS. A função *Watchdog* funciona como um mecanismo de comunicação intermediário entre API Gateway e o programa criado pelo usuário. Na função *Watchdog* existe um processador HTTP interno para realizar o procedimento de comunicação.

A figura 5.3 apresenta a visão geral da comunicação entre API *Gateway*, *Watchdog*

e programa da função. Por exemplo, o fluxo de execução de uma função que soma dois números ocorre da seguinte forma: uma requisição com dois números inteiros é enviada pelo usuário para o processador HTTP que processa a requisição, e envia os parâmetros para o programa da função através da entrada padrão. O programa da função realiza a operação soma de dois números no contêiner da função. Em seguida, o resultado da soma retorna para o processador HTTP que transforma esse resultado em uma resposta. Por fim, a resposta é enviada e recebida na interface do usuário.

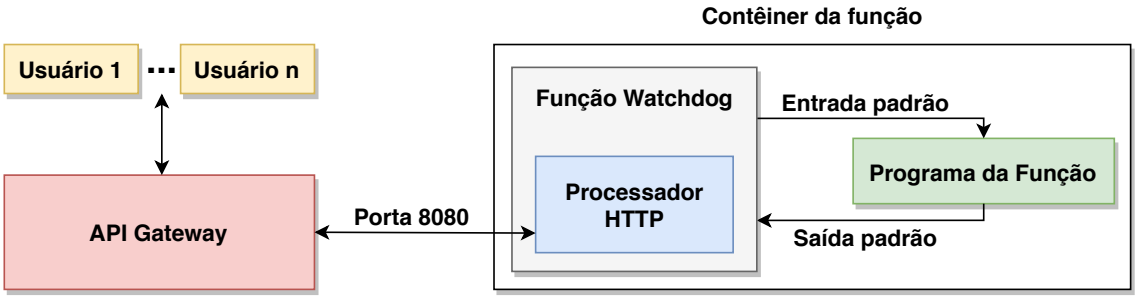



**Figura 5.3. Comunicação entre API *Gateway* e contêiner da função.**

O monitoramento de estatísticas no OpenFaaS ocorre através da ferramenta Prometheus, que é um programa mantido pela Linux Foundation [Linux Foundation 2016]. Prometheus foi adicionado ao OpenFaaS para monitorar serviços de acordo com a requisição das funções. Ele permite analisar funções do OpenFaaS produzindo informações relevantes, por exemplo, tempo médio de execução da função, número de requisições recebidas, e níveis de escalabilidade. A informação medida pelo componente é coletada toda vez que a requisição para uma função é detectada pelo API *Gateway*. OpenFaaS também suporta o uso de ferramentas de *dashboards* dinâmicos (por exemplo, Grafana [Grafana Labs 2014]) integradas com o Prometheus para visualização iterativa de informações em tempo de execução.

**Camada 2**: É onde ocorre a gerência dos contêineres através do componente denominado orquestrador. O orquestrador é responsável por configurar e coordenar contêineres na rede de forma automatizada. Contêineres podem ser alocados em servidores distintos pelo orquestrador de modo horizontal na rede, comunicando entre si como se estivessem lado a lado contidos no mesmo hardware. O OpenFaaS suporta dois tipos de orquestradores: Docker Swarm e Kubernetes.

Docker Swarm (ou Swarm) foi o primeiro orquestrador usado no OpenFaaS desde a primeira versão lançada em 2016. Swarm é um orquestrador simples, porém poderoso e nativo do Docker [Docker Inc. 2014]. Ele é recomendado para projetos simples e para usuários iniciantes no OpenFaaS. Swarm atua como uma ferramenta de clusterização que orquestra contêineres de acordo com a demanda provendo escalabilidade. Além disso, cria um cluster de contêineres utilizando apenas o comando `$ docker swarm init`. Através deste comando, a máquina hospedeira torna-se o nó principal do cluster. Novos nós podem ser adicionados com a alocação de um novo contêiner. Swarm é quem decide qual contêiner será alocado, simplificando o processo operacional de criação de novos nós do cluster de acordo com o recurso disponível no hospedeiro.

Kubernetes ou K8s foi criado pela Google [Google Inc. 2015] para gerenciar contêineres da rede interna da empresa. Em 2015, a empresa transformou o K8s em código aberto. A partir disso, o K8s passou a ser mantido pela Linux Foundation. Kubernetes cria ambientes *Serverless* com maior eficiência sobre contêineres. Além disso, apresenta algumas vantagens em relação ao Swarm, por exemplo, consumo menor de recursos do hospedeiro e da rede. Ele também permite implementar contêineres para uso local e na nuvem realizando pequenas alterações. Kubernetes é recomendado em projetos que demandam estabilidade, eficiência e baixo tempo de resposta. Entretanto, não existem restrições ao usar o Kubernetes em projetos mais simples. Em 2017, o OpenFaaS passou a ter suporte ao Kubernetes através da ferramenta faas-netes desenvolvida pelo criador da plataforma e disponibilizada em um repositório exclusivo [Ellis 2017a].

**Camada 1**: É o coração da plataforma OpenFaaS. O OpenFaaS utiliza contêineres para encapsular funções no formato de microsserviços. Nesta camada contêineres são criados através do Docker também denominado motor Docker. Docker é um sistema de virtualização que permite encapsular funções como serviços em contêineres. Contêineres são estruturas que possuem um sistema de arquivo próprio, executam processos, e contêm interfaces de rede. Eles compartilham recursos do *kernel* do sistema operacional hospedeiro com outros processos. Além disso, são rápidos e leves em relação à máquinas virtuais. Por exemplo, um contêiner construído a partir da imagem oficial do Linux Alpine [Alpine Linux 2018] gasta aproximadamente 3 segundos para inicializar e consome apenas 8 MB de espaço em disco.

#### 5.4.4.2. Funcionalidades

Nesta seção descrevemos as principais funcionalidades do OpenFaaS em relação às demais plataformas descritas neste minicurso. O OpenFaaS é uma plataforma de código aberto que possibilita a implementação de funções como serviço sobre contêineres totalmente escalável. O OpenFaaS fornece uma interface simples e amigável, onde funções são executadas usando poucos comandos ou via interface gráfica com apenas um clique do mouse. No OpenFaaS todo processo de instalação de serviços básicos, por exemplo, API Gateway e Prometheus ocorre executando o *script* `deploy_stack.sh`.

Funções *Serverless* criadas sobre OpenFaaS são totalmente independentes, sem um tipo de linguagem de programação estabelecida. No OpenFaaS o usuário pode definir qual linguagem de programação quer escrever suas funções, não estando preso a uma linguagem específica. Funções *Serverless* sobre o OpenFaaS também podem ser executadas em *clusters* gerenciados pelos orquestradores Swarm e K8s, como se estivessem no mesmo hardware. O OpenFaaS oferece suporte a plataforma OpenFaaS Cloud [Ellis 2017b], uma versão melhorada do OpenFaaS para gerência de projetos que contêm equipes de desenvolvimento grande. O OpenFaaS Cloud suporta Git e protocolo HTTPS para atender a demanda de datacenters, e está disponível em um repositório exclusivo desenvolvido pelo criador da plataforma junto à comunidade.

Funções assíncronas são funções que podem levar vários segundos para executar ou inicializar, e o usuário não precisa do resultado da função. Elas são interessantes em cenários como aprendizagem de máquina usando TensorFlow, requisições de tra-

balho em lote e limitação na taxa de funções Lambdas. OpenFaaS suporta processamento de funções assíncronas através de uma fila distribuída. A implementação desta fila é baseado no projeto NATS Streaming [Ellis 2017c], mas pode ser estendida para ser usado com Kafka [Ellis 2017d, Kafka 2011] ou qualquer outra estrutura similar a uma fila [Ellis 2017f].

### 5.4.5. OpenWhisk

A plataforma OpenWhisk [OpenWhisk 2016] é uma plataforma de computação sem servidor desenvolvida pela IBM e conta com suporte Apache. Ela é uma plataforma de código aberto e pode ser implantada sobre diferentes sistemas. OpenWhisk utiliza-se de outras plataformas que dão suporte à sua arquitetura, Docker [Docker Inc. 2008], Nginx [Igor Sysoev 2005], Kafka [Kafka 2011] e CouchDB [CouchDB 2005].

A plataforma pode ser invocada em uma instância local ou em algum provedor em nuvem. OpenWhisk possui uma interface cliente via linha de comando chamada “wsk"ela pode ser usada para criar, executar e gerenciar alguma instância OpenWhisk.

As linguagens suportadas pela plataforma, até o momento de escrita desse minicurso são JavaScript[3], Go, Python, Java, PHP, Ruby, Swift e .NET. O modelo de programação OpenWhisk é baseado em três pontos: Ações, Gatilhos e Regras. As Ações são as funções *Serverless*, também conhecidas e citadas nesse minicurso como Lambdas. Os Gatilhos são os disparados sempre que um evento ocorra, um evento pode ter várias origens, por exemplo uma mensagem chegando em uma fila de mensagens ou a chegada de dados de um dispositivo IoT. As Regras associam um Gatilho a uma Ação.

#### 5.4.5.1. Arquitetura

A figura 5.4 apresenta o fluxo de trabalho da plataforma OpenWhisk. Primeiro há uma solicitação HTTP, a aplicação *wsk* traduz os comandos para requisições HTTP para o sistema OpenWhisk.

O ponto de entrada da plataforma é a plataforma Nginx. O Nginx é um servidor web orientado a eventos. Lançado em 2004, ele é responsável por uma grande parcela do tráfego global, considerando apenas sites ativos. Após processar a solicitação e, não havendo mais nada as ser feito pelo Nginx, a solicitação é repassada ao controlador. O controlador é uma aplicação que serve como interface para as solicitações do usuário. O controlador traduz uma solicitação do usuário para uma ação existente no sistema. Após a tradução o controlador autentica as permissões do usuário que está fazendo a solicitação. A autenticação e autorização é feita através de uma instância do banco de dados CouchDB. Após a validação de que o usuário tem os privilégios necessários o controlador consulta novamente o banco de dados carregando a ação. O registro da ação contém os parâmetros que deseja-se transmitir a ação, o código a ser executado na ação, além de conter as restrições de recursos do sistema que serão detalhados na seção 5.4.5.2.

O controlador possui uma visão geral do sistema e fica constantemente verificando o estado do sistema, possui também um balanceador de carga e através dele escolhe um

[3]Através do interpretador Node.js

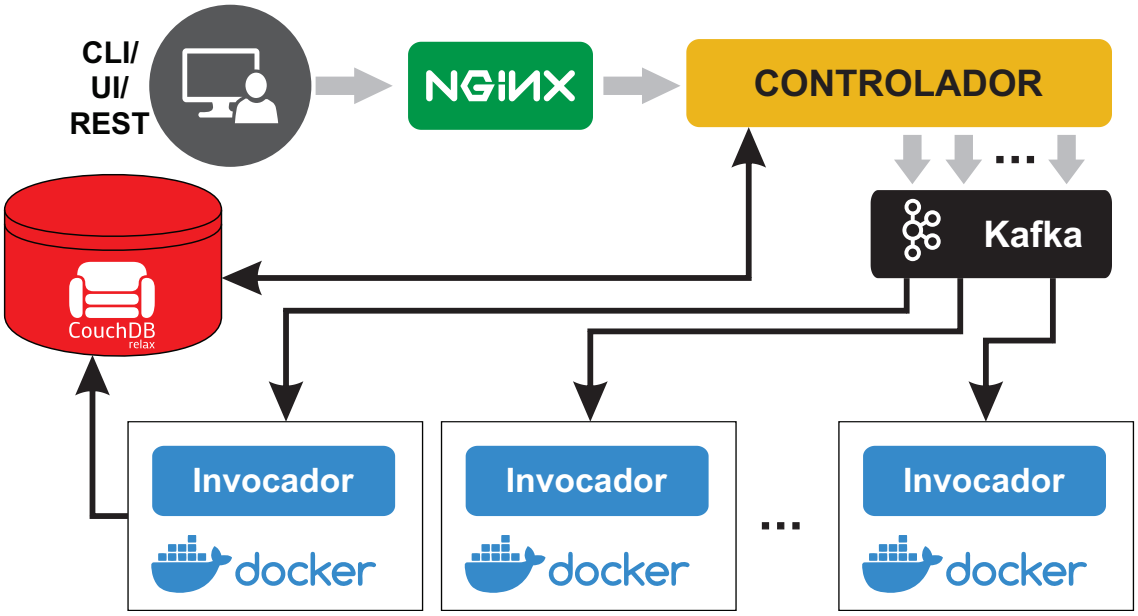



**Figura 5.4. Arquitetura do OpenWhisk.**

dos invocadores disponíveis para invocar a ação solicitada. Exitem dois problemas, mais graves, que podem ocorrer nesse ponto. O primeiro deles é o sistema falhar perdendo a invocação, e o segundo, o sistema está com uma demanda tão alta que precisa esperar que outras chamadas sejam concluídas. A solução usada para solucionar ambos é um serviço de mensagens chamado Kafka. Ele é um sistema de mensagens de alto rendimento, basicamente ele desacopla os produtores de fluxo dos consumidores de fluxo garantindo através de persistência a entrega das mensagens, no caso do OpenWhisk ele desacopla o controlador (Produtor) dos invocadores (Consumidor) e toda a comunicação entre o controlador e os invocadores é feita utilizando o Kafka.

O invocador tem como objetivo invocar uma ação. Para executar ações o invocador utiliza o Docker para configurar e gerar um contêiner. Para cada ação chamada é criado um contêiner para ela, o código da ação é inserido dentro do contêiner. O contêiner, então, é executado usando os parâmetros passados, o resultado da computação é obtido pelo invocador e o contêiner é destruído. Os resultados obtidos pelo invocador são salvos no banco de dados de ativações, eles residem no CouchDB. O registro das invocações e dos resultados podem ser acessados.

### 5.4.5.2. Detalhes do Sistema

Nesta sessão serão descritos alguns detalhes de sistema da plataforma OpenWhisk. As Ações, Gatilhos e Regras pertencem, sempre, a um *namespace*[4] e, opcionalmente a um *package*. Um *package* pode conter ações. Entretanto não se pode fazer aninhamento de *packages*, ou seja, um *packages* não pode conter outro.

As Ações possuem alguns limites, como por exemplo, uso de memória e tempo

[4]Optou-se por manter os mesmos nomes usados na plataforma.

**Tabela 5.2. Parâmetros de sistema da plataforma OpenWhisk.**

| Limite | Descrição | Configurável | Unidade | Padrão |
|---|---|---|---|---|
| *timeout* | Um contêiner não está autorizado a executar mais do que N milisegundos | por Ação | ms | 60000 |
| *memory* | Um contêiner não está autorizado a alocar mais do que N Megabytes | por Ação | MB | 256 |
| *logs* | Um contêiner não está autorizado a escrever mais do que N Megabytes na saída padrão | por Ação | MB | 10 |
| *concurrent* | Não pode ser enviado mais do que N ativações por *namespace* em execução ou na fila para execução | *namespace* | número | 100 |
| *minuteRate* | Não pode ser enviado mais do que N ativações por *namespace* por minuto | *namespace* | número | 120 |
| *codeSize* | Tamanho máximo que o código da Ação pode ter | por ação | MB | 48 |
| *parameters* | Tamanho máximo dos parâmetros que podem ser inseridos | não configurável limite por Ação/*package*/Gatilho | MB | 1 |
| *result* | Tamanho máximo do resultado de uma Ação | não configurável limite por Ação | MB | 1 |

máximo de computação. Esses limites podem ser configurados e cada limite possui uma granularidade diferente. A tabela 5.2 mostra os limites que podem ser configurados e o nível que pode ser configurado se é por Ação, por *namespace* ou por *package*. Quevedo et al. [Quevedo et al. 2019] avaliaram o desempenho da plataforma para as linguagens Java e JavaScript. Eles avaliaram a plataforma utilizando os limites padrão e o que eles consideraram a melhor configuração para a plataforma.

A Plataforma OpenWhisk conta com vasta documentação e, como já foi dito, com suporte Apache. Além disso, conta com diversos trabalhos acadêmicos que avaliaram ou usaram a plataforma em suas pesquisas.

### 5.4.6. OpenLambda

A plataforma de computação sem servidor OpenLambda [Hendrickson et al. 2016], desenvolvida em 2016, teve sua implementação norteada pela plataforma *Serverless* da Amazon. Ela foi escrita em linguagem Go e conta com licença da Apache [Apache 2004].

A documentação para esta plataforma é escassa. O foco do projeto é criar uma plataforma com a abordagem em computação sem servidor. A plataforma não faz apresentação de sua arquitetura, nem todas as linguagens suportadas pela plataforma. Entretanto, pode-se constatar que a plataforma aceita linguagens de programação interpretativas de compilação *Just-in-time*. As linguagens, até o momento de escrita desse minicurso são Java, JavaScript e Python.

### 5.4.7. Comparativo

As plataformas de computação *Serverless* apresentadas neste minicurso podem ser diferenciadas entre plataformas comerciais (AWS Lambda, Azure Functions e Cloud Functions) e de código aberto (OpenFaaS, OpenWhisk e OpenLambda). Entretanto, algumas das plataformas de código aberto oferecem funções como serviço em seus servidores. São elas OpenFaaS e OpenWhisk.

A AWS Lambda, pioneira no paradigma FaaS, tem como forte atrativo a estreita integração com outros serviços *cloud* oferecidos pela Amazon. A plataforma Azure Functions, também conta com fácil integração com outros serviços em nuvem da Microsoft Azure. Além disso, Azure permite que as funções sejam testadas localmente antes de serem carregadas para a plataforma. O Cloud Functions se diferencia dos demais por configuração personalizada para IoT através do serviço de mensagens da Google globalmente distribuído. O Cloud Functions pode ser facilmente integrado a serviços de terceiros.

As plataformas de código aberto apresentadas neste minicurso representam soluções para pesquisas e utilização de computação sem servidor. A plataforma OpenFaaS rapidamente ganhou destaque. Pode-se destacar como um grande atrativo para a adoção da plataforma é a grande quantidade de linguagens de programação oficialmente aceitas pela plataforma. Além disso, ela conta com suporte para que o usuário acrescente alguma linguagem que queira usar. Graças ao sucesso que a plataforma obteve a documentação é ampla e bastante completa.

A plataforma OpenWhisk tem como atrativo o suporte da Apache e faz parte da plataforma *cloud* da IBM. Além disso, conta com ampla gama de linguagens de programação. A OpenWhisk, assim como a OpenFaaS, dá suporte a inserir novas linguagens de programação. Entretanto, para isso é necessário uma ferramenta diferente do “wsk", o que pode tornar a inserção um pouco mais complexa.

A tabela 5.3 apresenta um resumo das plataformas mostrando as linguagens suportadas e outras informações necessárias.

## 5.5. Projetos de pesquisa

Nesta seção descrevemos tópicos de pesquisa *Serverless* de seis áreas da computação. Os tópicos abordados nesta seção foram baseados em artigos publicados no período de 2017 a 2019.

### 5.5.1. Processamento de vídeo

Processamento de vídeo na Internet aumentou de forma exponencial após a popularidade de provedores de serviço *streaming* como YouTube e Netflix. Ainda assim, alguns desafios em processar vídeo de forma escalável com tempo de resposta em poucos segundos estão em aberto. Processamento de vídeo consome várias CPUs, por exemplo, um vídeo no formato 4K com uma hora de vídeo leva mais de 30 horas de uso de CPUs para processar. Para atender o tempo de resposta em poucos segundos sem atrasos, múltiplos *threads* devem ser invocados funcionando em paralelo com múltiplas CPUs. Além disso, alguns tipos de vídeos contêm codificadores que não possibilitam processar o vídeo de forma paralela em milhões de partes. *Serverless* apresenta características que enquadram

**Tabela 5.3. Tabela comparativa de plataformas de computação sem servidor.**

| Plataforma | Linguagens Suportadas | Infraestrutura | Virtualização | Gatilhos | Tempo máximo de computação | Cobrança |
|---|---|---|---|---|---|---|
| **Amazon Lambda** | C#, Go, Java, Powershell, Ruby, Python, Node.js | Nuvem | Firecracker (KVM) | HTTP, serviços da AWS | 900 | Requisição, tempo de execução memória |
| **Azure Functions** | C#, F#, Java, Python, JavaScript | Nuvem, Local | Imagens de SO | HTTP, serviços da Azure | 600 | Requisição, tempo de execução |
| **Cloud Functions** | BASH, Go, Node.js, Python | Nuvem, Local | Não definido | HTTP, Pub/Sub, Google storage | 450 | Requisição, tempo de execução memória |
| **OpenFaaS** | Go, C#, JavaScript, Java, Ruby e PHP | Nuvem, Local | Docker | HTTP, FaaS-CLI | Indefinido | Cloud: Não definido Local: Não se aplica |
| **OpenWhisk** | JavaScript, Go, Python, Java, PHP, Ruby, Swift e .NET | Nuvem, Local | Docker | HTTP, IBM Cloud, wsk | 300 | IBM Cloud: Requisição, tempo de execução Local: Não se aplica |
| **OpenLambda** | Java, JavaScript e Python | Nuvem, Local | Docker | HTTP | Indefinido | Não se aplica |

no contexto de processamento de vídeo como escalabilidade, paralelismo, e processamento em poucos segundos. No entanto, segundo Fouladi et al. [Fouladi et al. 2017], alguns pontos como desempenho e custos no processamento de vídeo usando *Serverless* não estão claros.

Zhang et al. [Zhang et al. 2019] apresentam um estudo sobre os desafios do processamento de vídeo utilizando computação *Serverless*. Neste trabalho esquemas de implementação e configuração de funções padrão utilizadas em processamento de vídeo foram implementados e avaliadas em duas plataformas *Serverless*, AWS Lambda e GCF (*Google Cloud function*). Para realizar esta análise foram utilizadas as métricas de medição de impacto da alocação de recursos de funções referente ao consumo de memória, esquemas de implementação de funções locais ou APIs externas, comportamento do vídeo em relação à duração, e o custo monetário.

Como resultado do artigo, os autores destacam que implementar funções de processamento de vídeo não é uma tarefa trivial porque as funções estão limitadas a quantidade de recurso disponível na plataforma, por exemplo, tamanho da memória. Aumentar o tamanho da memória não é uma estratégia viável porque não melhora o desempenho de processamento, mesmo podendo configurar dinamicamente o tamanho da memória de acordo com a demanda da aplicação. Em relação ao custo, tarefas de processamento de vídeo complexas executadas através de API externas são mais caras do que funções implementadas localmente. Por exemplo, realizar detecção de face em um vídeo com o modelo parcialmente treinado localmente tem custo menor em relação à APIs externas com o mesmo resultado. Como último resultado, os autores destacam que o AWS Lambda tem mais vantagens que o GCF em termos de duração de execução e custo monetário utilizando o mesmo esquema de configuração. No entanto, AWS Lambda tem o desempenho afetado devido à restrições de recurso, e não é estável como o GCF.

Ao et al. [Ao et al. 2018] projetaram o Sprocket, um framework *Serverless* para processamento de vídeo. No Sprocket, usuários podem executar um conjunto de operações em paralelo no conteúdo do vídeo de modo modular, criando *pipelines* de processamento. Sprocket processa o vídeo de acordo com a função carregada pelo usuário. O usuário implementa as funções usando linguagem de domínio específico, como um grafo de fluxo de dados acíclico direcionado (GAD). O GAD é composto por vértices responsáveis por executar funções individuais, definidas pelo usuário. As funções são executadas nos dados (quadros individuais do vídeo, grupos de quadros ou segmentos compactados de quadros consecutivos) que chegam no grafo. Após ser processado, o dado é encaminhado para o próximo vértice até chegar nos vértices borda do grafo denominados vértices de saída. No GAD, arestas são denominadas fluxos, e tem a funcionalidade de transmitir dados entre vértices. Um programa GAD é representado como um *pipeline*, tal que, os vértices no DAG representam os estágios do *pipeline* de processamento.

Com Sprocket funções complexas de processamento de vídeo podem ser implementadas facilmente devido ao projeto do framework. Sprocket foi implementado nas plataformas AWS Lamda, Microsoft Azure Function e Google Cloud Function. Como resultado, os autores avaliaram uma variedade de condições para mostrar que o sistema consegue ter paralelismo, baixa latência e baixo custo. No artigo, os autores mencionam que um vídeo de 3.600 segundos custa menos de três dólares por hora de vídeo processado

usando o Sprocket sobre as plataformas *Serverless* avaliadas.

### 5.5.2. Aprendizagem de máquina

Modelos de aprendizagem de máquina são treinados em *clusters* compostos por máquinas virtuais utilizando em seu fluxo de trabalho processos como pré-processamento, modelo de treinamento e ajuste de hiperparâmetros. Cada processo de treinamento requer uma quantidade diferente de recurso gerando sobrecarga ou subutilização do *cluster*. *Serverless* pode contornar estes desafios escalonando cada processo de treinamento independente da demanda de recurso [Jonas et al. 2019].

Ishakian et al. [Ishakian et al. 2018] avaliaram o desempenho de modelos de rede neural sobre a plataforma AWS Lambda usando o framework de aprendizagem profundo Amazon MXNet. Na avaliação realizada, três modelos de reconhecimento de imagem (SqueezeNet, ResNet e ResNeXt-50) foram testados. Para avaliar estes modelos dois cenários foram monitorados: (i) se o contêiner precisa ser inicializado e se a função Lambda está no estado de execução e termina de ser processada. Isso implica em uma sobrecarga adicional no sistema e maior atraso; (ii) se o contêiner é inicializado e a função está no estado de execução. A sobrecarga de execução é a mesma da função Lambda. As métricas consideradas nestas situações foram o tempo de resposta, tempo de predição e custo da função. Os resultados apresentados demonstraram que, no primeiro cenário, a latência é aceitável, enquanto no segundo cenário ocorre uma sobrecarga significativa.

Feng et al. [Feng et al. 2018] investigaram o treinamento de redes neurais utilizando paralelismo de dados sobre a plataforma AWS Lambda. Neste trabalho foram implementados técnicas de otimização para reduzir latência, custo de desempenho e monetário do processamento de redes neurais utilizando *Serverless*. Os experimentos foram realizados baseado em dois conjuntos de dados CIFAR-10 e MNIST. CIFAR-10 foi treinado por uma rede neural de convolução composta por duas camadas de convolução, duas camadas de *pooling*, duas camadas de normalização, duas camadas totalmente conectadas e uma camada de saída *softmax*. MNIST foi treinado por uma rede neural totalmente conectada, cuja estrutura é investigada através do ajuste do hiperparâmetro. Ambos os conjuntos de dados foram gerados aleatoriamente com um milhão de amostras sendo executadas na plataforma AWS Lambda. Fatores como latência, uso de memória e custo monetário foram monitorados. Os resultados obtidos demonstraram vantagens no desempenho e custo do treinamento de redes neurais sobre a plataforma AWS Lambda.

### 5.5.3. Computação Científica

Operações com álgebra linear são utilizadas em problemas de computação científica como simulação climática, montagem de genomas e dinâmicas dos fluídos. Esses problemas requerem ser processados em supercomputadores ou *clusters* de alto desempenho. Infelizmente, executar operações com álgebra linear de modo distribuído em grande escala é um desafio para cientistas e analistas de dados devido à restrições de acessibilidade e gerenciamento do *cluster*. *Serverless* fornece acesso à plataformas com grande capacidade de processamento, sem o usuário se preocupar com gerenciamento do *cluster*. Ainda assim, plataformas *Serverless* têm recurso limitado ou subutilizado com paralelismo variando drasticamente ao processar problemas científicos, como apontado por Jo-

nas et al. [Jonas et al. 2019].

Shankar et al. [Shankar et al. 2018] criaram Numpywren, um sistema distribuído *Serverless* que permite executar algoritmos de álgebra linear em larga escala utilizando funções *Serverless*. Numpywren contém uma linguagem de domínio específico denominada LAmbdaPACK. LAmbdaPACK permite usuários desenvolverem algoritmos de álgebra linear como multiplicação de matrizes, decomposição de valor único e decomposição de Cholesky sobre a plataforma AWS Lambda. Todos esses algoritmos tem complexidade cúbica e podem gastar horas para serem processados. Sobre o processamento, Numpywren consegue se adaptar a quantidade de paralelismo disponível e decompor programas em várias CPUs, provendo usabilidade e tolerância à falhas. Numpywren foi comparado ao ScalaPACK (biblioteca FORTRAN industrial) e ao DASK (biblioteca de tolerância à falhas escrita em Python). Os resultados demonstraram que o Numpywren comparado ao ScaLAPACK consome 20 a 30% menos horas de CPU. Em relação a eficiência, para problemas grandes Numpywren é 320% mais rápido que DASK.

Werner et al. [Werner et al. 2018] desenvolveram um protótipo de sistema *Serverless* para multiplicação de matrizes. Para realizar multiplicação de matrizes o sistema precisa de três funções sem estado para obter o resultado geral. Nenhuma dessas funções se comunica entre si, sendo os dados transferidos para nuvem. O sistema é composto por três elementos denominados cálculo, armazenamento, e orquestração. O elemento cálculo processa essas funções. O elemento armazenamento fica responsável por armazenar os dados processados na nuvem. Por fim, o elemento orquestração garante que o processamento ocorra de modo distribuído e compartilhado entre as três funções. O projeto do sistema foi implementado sobre a plataforma AWS Lambda. As funções Lambda foram escritas na linguagem Python porque a plataforma AWS fornece bibliotecas para manipular dados de matrizes e operações com multiplicação. Para validar o sistema três fatores foram avaliados: escalabilidade, capacidade de ajuste e custo de desempenho comparados aos *clusters* Apache Spark11 e Hadoop MapReduce12 gerenciados pelo serviço Amazon EMR13. Os resultados obtidos demonstraram que o sistema tem custos de infraestrutura e operacional baixos se comparados à outras plataformas.

### 5.5.4. Computação nas bordas

É um paradigma emergente que ganhou atenção da indústria e academia devido à popularidade de tecnologias como 5G, Internet das coisas (IoT) e redes veiculares. Nesse paradigma serviços de computação em nuvem foram estendidos para borda da rede para prover baixa latência, tempo de resposta rápido e mobilidade Khan et al. [Khan et al. 2019]. *Serverless* originalmente foi criado para computação em nuvem, mas tornou-se primordial para computação na borda. Computação na borda apresenta limitações como gerenciamento, produção de grande volume de dados, e alto custo. *Serverless* pode solucionar essas limitações, no entanto, as arquiteturas de plataformas *Serverless* precisam ser adaptadas para suportar computação de dispositivos na borda. Além disso, questões de pesquisa como desempenho, elasticidade, gerenciamento e segurança entre *Serverless* e computação na borda precisam ser reavaliados [Glikson et al. 2017].

Xiong et al. [Xiong et al. 2018] implementaram KubeEdge, uma plataforma construída sobre Kubernetes que permite funções *Serverless* serem executadas sobre nós borda

da rede e servidores de nuvem, em tempo de execução, de modo unificado. KubeEdge utiliza contêineres orquestrados pelo Kubernetes fornecendo um canal de comunicação entre nós borda e servidores de nuvem via protocolo RPC (*Remote Procedure Call*). KubeEdge suporta mecanismos de sincronização e armazenamento de metadados com gerenciamento automatizado de funções sobre contêineres. KubeEdge é composto por quatro componentes: barramento Kube, controlador nó borda, serviço de sincronização de metadados, e núcleo nó borda. Barramento Kube é uma camada de rede virtual que conecta nós borda e máquinas virtuais alocadas na nuvem. Controlador nó borda é um *plugin* do controlador Kubernetes responsável por gerenciar remotamente nós borda, permitindo que serviços sejam implantados via Kubernetes. Serviço de sincronização de metadados ocorre de modo bidirecional entre a borda da rede e nuvem. Por fim, o núcleo nó borda é composto por um agente que roda sobre nós borda. O agente é responsável por inicializar e gerenciar contêineres e funções. No artigo os autores não apresentaram nenhum resultado de desempenho da plataforma.

Vilalta et al. [Vilalta et al. 2017] propuseram TelcoFog, uma arquitetura de computação distribuída que suporta serviços como 5G, NFV e IoT na extremidade da rede integrado com servidores de nuvem. TelcoFog é composto por nós na extremidade da rede, um controlador centralizado e serviços. Nós são integrados à infraestrutura de telecomunicações e podem trabalhar em uma rede sobreposta à rede de operadoras permitindo a integração de serviços. O controlador TelcoFog é responsável por garantir a qualidade dos serviços baseado na modelagem de dados de serviço usando a linguagem YANG (*Yet Another Next Generation*). YANG é uma linguagem de modelagem de dados integrada a arquitetura de gerenciamento e orquestração do operador usada para modelar operações e configuração de dados dos dispositivos da rede. Os serviços rodam sobre a infraestrutura de telecomunicações e o TelcoFog. TelcoFog permite que vários serviços sejam implantados dinamicamente de modo distribuído com baixa latência para operadores. O protótipo foi validado por meio de uma prova de conceito para serviços de IoT. Os resultados apresentados demonstram o tempo gasto em cada componente da arquitetura.

### 5.5.5. Segurança

Funções *Serverless* são encapsuladas em contêineres. Contêineres são construídos via imagens Docker composta por softwares de desenvolvedores oficiais e da comunidade. Softwares adicionados a imagem Docker podem conter vulnerabilidades de segurança, e essas vulnerabilidades podem comprometer a segurança de dados ou a integridade do sistema. *Hackers* podem utilizar o processamento distribuído *Serverless* para executar sistemas maliciosos para realizarem ataques. Esses sistemas são difíceis de serem detectados devido à estrutura da arquitetura *Serverless* [Wu et al. 2018].

Bila et al. [Bila et al. 2017] propuseram uma arquitetura automatizada para detecção de vulnerabilidades em contêineres utilizando OpenWhisk e Kubernetes. Neste artigo os autores estenderam a API do Kubernetes para criar um mecanismo de quarentena capaz de lidar com vulnerabilidades em imagens Docker e contêineres isolando ameaças do restante da rede em tempo de execução. Além disso, criaram um gerenciador de políticas baseado no OpenWhisk que permite o usuário adicionar regras para melhorar o sistema de isolamento de contêineres. Essas regras recebem informações do mecanismo de detecção de vulnerabilidades e acionam a API do Kubernetes para realizar ações de segurança.

Outra contribuição do trabalho refere-se a estender um serviço de verificação de vulnerabilidades para gerar eventos que ativam as regras do gerenciador de políticas para ajudar na detecção de ameaças. A abordagem proposta fornece varredura contínua de contêineres. Após uma ameaça ser detecta, ela é encaminhada para uma estrutura de isolamento que analisa relatórios de vulnerabilidade, e toma decisões com base em políticas definidas pelo usuário. Nenhum resultado foi apresentado para validar a arquitetura. Como trabalhos futuros, os autores pretendem utilizar inteligência artificial para gerar políticas de segurança de forma automatizada de acordo com o tipo de ameaça detectada.

Wu et al. [Wu et al. 2018] propuseram SLBot, um sistema *Serverless* baseado no *Service Flux*, um protocolo de comunicação com três canais de transmissão de dados que garante maiores níveis de disfarce da aplicação na rede. Neste trabalho os autores também discutem a viabilidade e eficácia do uso de serviços públicos na Web para a construir *botnets Serverless*. Testes com SLbot foram realizados comparando o SLbot a outras tecnologias similares. Os resultados obtidos demonstraram que o SLBot é mais eficiente que *botnets* tradicionais, sendo mais seguro. Como conclusão do artigo, os autores apresentaram técnicas para detectar e combater *botnets* construídas de forma semelhante ao SLBot.

### 5.5.6. Processsamento em SmartNICs

Todas as plataformas de computação em nuvem realizam processamento *Serverless* sobre servidores com CPU x86_x64 utilizando virtualização de funções Lambda de modo paralelo. CPUs x86_x64 foram projetadas para processar uma sequência de instruções de modo rápido. No entanto, são inadequadas para executar milhões de funções em paralelo. Cada função interrompe a CPU para armazenar estados em registradores ou memória, gerando atrasos de processamento e sobrecarga da memória devido à troca de contexto. Para contornar essas limitações, provedores de computação em nuvem com suporte *Serverless* estão adotando o uso de SmartNics na tentativa de melhorar a eficiência energética, redução de custos operacionais, e latência de *datacenters*.

Liu et al. [Liu et al. 2019] propuseram uma plataforma distribuída que permite executar microsserviços em *datacenters* usando SmartNics denominado E3. E3 processa microsserviços como processos com múltiplos threads sobre SmartNics. O projeto do E3 estende os componentes chave da plataforma *Azure Service Fabric* sobre SmartNics instaladas em servidores agrupados em *racks*. Cada *rack* contém um comutador ToR (*Top-of-Rack*), onde as SmartNics são conectadas. Para detectar a sobrecarga de processamento, o E3 usa técnicas como balanceamento de carga com roteamento de caminhos múltiplos com custo igual entre placa para o hospedeiro, posicionamento da localização onde o microsserviço será processado utilizando o reconhecimento da topologia da rede, e um orquestrador no plano de dados.

Sobre os componentes do E3, ele contém um controlador responsável por gerenciar o recurso do *cluster*, e aplicação *runtime* do microsserviço em cada hospedeiro e SmartNic. Cada *runtime* inclui um motor de execução, um agente orquestrador e um subsistema de comunicação. O E3 segue o modelo de programação baseado no fluxo de dados representado por um grafo acíclico direcionado (GAD). GAD é representado com microsserviços sendo os nós do grafo, e as arestas os canais de comunicação na direção

do fluxo RPC (*Remote procedure call*). Vários GADs podem ser criados e executados simultaneamente no E3. GAD descreve todos os caminhos do RPC e execução de um único microsserviço. O E3 foi implementado dentro de um *cluster* composto por servidores Xeon com até 4 SmartNics de 10 Gbps por servidor. Os resultados obtidos do E3 demonstram que *offloading* de microsserviços sobre SmartNics melhoram a eficiência energética em até 3x, e reduzem a latência em até 4x comparado com outras plataformas.

Choi et al. [Choi et al. 2019] apresentaram $\lambda$-NIC, um *framework Serverless* que processa milhões de funções Lambdas de modo iterativo em uma SmartNic composta por várias NPUs (*Network Processing Unit*). $\lambda$-NIC introduz uma nova abstração denominado casamento-Lambda que oculta a complexidade existente em SmartNics. Além disso, estende o modelo de máquina casamento-ação do P4 para melhorar a eficiência de códigos e executar funções Lambdas sobre SmartNics.

No $\lambda$-NIC, funções Lambdas são compiladas e executadas em tempo de execução com a chegada de pacotes na placa. Quando um pacote chega, o cabeçalho do pacote é analisado. Se ocorre o casamento do pacote com a função carregada no $\lambda$-NIC, uma ação é realizada. $\lambda$-NIC infere quais cabeçalhos do pacote são usados por cada função e gera automaticamente um analisador correspondente para os cabeçalhos, eliminando a necessidade de especificar lógica de processamento de pacotes manualmente para cada função. Usuários programam funções sobre no $\lambda$-NIC sem importar com detalhes em nível de hardware, por exemplo, $\lambda$-NIC analisa padrões de acesso à memória e endereços de acesso válido. O $\lambda$-NIC mapeia funções de modo otimizado nas memórias da SmartNic, garantindo que os acessos à memória sejam isolados. Funções são alocadas para processamento em uma única NPU sendo executada até ser concluída.

Por fim, $\lambda$-NIC emprega uma semântica de entrega pouco consistente com RDMA (*Remote Direct Memory Access*) processando solicitações diretamente nos núcleos da placa sem utilizar a CPU do hospedeiro. Os resultados obtidos mostram que $\lambda$-NIC alcança uma melhora de 736x a 880x na latência e vazão, reduzindo o uso de CPU e memória em relação a outros *frameworks Serverless* existentes.

Pacífico et al. [Pacífico et al. 2020] projetaram um sistema de processamento de pacotes *Serverless* com OpenFaaS orquestrando funções via contêineres e realizando *offloading* na plataforma NetFPGA SUME 10 G. Os principais componentes que compõem o sistema são a plataforma OpenFaaS, agente e processador de funções *Serverless*. Antes de executar uma função, o usuário precisa adicionar o código da função no OpenFaaS para enviar requisições de execução da função via interface gráfica ou linha de comando. Quando uma requisição chega no OpenFaaS, um contêiner é alocado para função. Após a função ter sido criada, uma requisição de execução com o tempo de processamento e função eBPF são enviados para o componente agente. O agente é responsável por escalonar a execução de funções de acordo com disponibilidade do processador. A computação da função termina com o usuário recebendo o resultado de processamento, liberando processador e contêiner.

Para melhorar o desempenho do sistema otimizações de hardware foram realizadas, como processador eBPF com *pipeline*, sistema de memória com *buffer* duplo, estrutura FIFO_MD para cópia zero de pacotes, e uso de mapas. Os autores avaliaram fatores do sistema como processamento *Serverless*, tempo de inatividade, vazão, latência

e energia. Quatro funções de rede foram implementadas: Wire, learning switch L2, roteador IPv4 e mitigação DDoS. Os resultados do sistema mostram que o sistema opera em taxa de linha de modo programável via espaço de usuário consumindo menos energia em relação a outros sistemas existentes.

## 5.6. Desafios e limitações

Essa seção é dedicada a apresentar e discutir os desafios das tecnologias *serverless* assim como suas limitações. Exemplos de aplicações, tópicos de pesquisa e aprendizados serão apresentados.

A computação *Serverless* obteve muito sucesso em sua aplicabilidade em diversas classes de cargas de trabalho como desenvolvimento de APIs e tratamento de eventos em tempo real como descrito por Jonas [Jonas et al. 2019].

No entanto, existem alguns tópicos em que as ofertas de computação *Serverless* são limitadas. Uma delas seria ser eficiente no processamento de dados. Outra seria que ela dificulta o progresso no desenvolvimento de sistemas distribuídos conforme apresentado por Hellerstein [Hellerstein et al. 2018].

Alguns exemplos de limitações ofertadas pela computação *Serverless* são apresentadas por Hellerstein [Hellerstein et al. 2018] e podem ser vistas a seguir:

- **Ciclo de vida limitado** No caso da AWS por exemplo, após 15 minutos, funções invocadas são desligadas pela infraestrutura *Lambda*.
- **Gargalos de Entrada/Saída (I/O)** A infraestrutura *AWS Lambda* conecta-se e depende de outros serviços da nuvem como dispositivos de armazenamento conectados com acesso via rede. Na prática, isso significa transferir dados entre equipamentos dentro dos centros de dados.
- **Comunicação entre dispositivos** Enquanto a *AWS Lambda* conecta-se via rede, elas não podem ser diretamente endereçáveis na rede enquanto estão em execução.
- **Ausência de computadores especializados** Atualmente as ofertas de computação *Serverless* são baseadas em seções de tempo de uso de uma CPU e alguma quantidade de memória primária. Não se utiliza computadores especializados no formato de computação *Serverless*

Jonas [Jonas et al. 2019] apresenta alguns exemplos práticos de aplicações limitadas pela computação *Serverless*:

- **Codificação de vídeo em tempo real** Serviços de armazenamento muito lentos para suportar comunicação com granularidade muito fina entre funções. Soluções paralelas porém diretas tem melhor desempenho neste tipo de tarefa.
- **MapReduce** O embaralhamento necessário na implementação de MapReduce não escala em função da latência na busca de objetos e por limites de velocidade nas operações de leitura/escrita por segundo.

- **Computar modelos de Álgebra Linear** Difícil de implementar comunicação difusa (*broadcast*) devido a limitação na latência de dispositivos de armazenamento. Além disso, provisionar um conjunto fixo de funções pode deixar recursos com sub-utilização.

- **Aprendizado de máquina** Ao construir encadeamento (*pipeline*) a falta de armazenamento de acesso veloz para armazenar parâmetros impacta no desempenho das tarefas de agregação de dados nesses tipos de algoritmos.

- **Bancos de dados** A falta de memória compartilhada impede o desenvolvimento de bancos de dados, pois o desempenho é baixo em função da latência para acessar serviços de armazenamento em blocos remotos.

Castro [Castro et al. 2019] afirma que: uma vez que a computação *Serverless* é uma área nova, existem diversas oportunidades de pesquisa a serem endereçadas e que representam desafios em Ciência da Computação.

O trabalho de Jonas [Jonas et al. 2019] descreve pesquisas desenvolvidos pela Universidade de *Berkeley* enfatizam: centros de dados, sistemas distribuídos, aprendizado de máquina e modelos de programação.

Como uma visão geral, são citados por Jonas [Jonas et al. 2019] cinco categorias dessas áreas que representam desafios:

- Abstrações;
- Sistemas;
- Redes;
- Segurança;
- Arquitetura.

Abaixo são listadas alguns exemplos de áreas de pesquisa que se enquadram dentro das categorias citadas:

- **Abstrações** Hoje as ofertas em computação *Serverless* permitem especificar a quantidade de memória e tempo de execução aos quais as funções irão fazer uso. No entanto, outros recursos não estão sob controle do desenvolvedor.

- **Dependência de dados** Atualmente as funções não tem conhecimento sobre a dependência de dados entre funções. Essa ignorância gera comunicações ineficientes por transações feitas com desempenho sub-ótimo.

- **Persistência** Existe uma carência em soluções para computação *Serverless* que implementem armazenamento persistente e efêmero por funções.

- **Tempo de inicialização** Existem três etapas na inicialização de funções: (1) agendamento e inicialização de ambiente da linguagem utilizada na função (*runtime*). (2) Descarregar (*Download*) o código, as bibliotecas e recursos necessário para executar a função. (3) Carregar bibliotecas e estruturas de dados em memória para que estejam disponível quando a função for executada.
- **Redes e sistemas distribuídos** Permitir que funções sejam colocadas em servidores específicos como por exemplo escolher a localidade por proximidade. Colocar funções em uma mesma máquina virtual para poder comunicar com mais eficiência.
- **Segurança** Agendamento aleatório para isolamento físico de funções. Além disso as funções deveriam ter acesso granular de configurações, chaves privadas, objetos de armazenamento e até dados temporários localmente.
- **Arquitetura de computadores** Equipamentos (*hardware*) heterogêneos, precificação e fácil administração para que usuários da computação *Serverless* tenham total controle do desempenho das funções.

Uma predição feita por Jonas [Jonas et al. 2019] é que a computação *Serverless* irá decolar e que a nuvem será híbrida. As aplicações irão diminuir em tamanho mesmo havendo alguns casos em que, por motivos de regulação e governança de dados, as aplicações precisarem usar legados.

Como mostrado por Ao [Ao et al. 2018], processamento de vídeo é um tópico emergente na utilização de computação *Serverless*. Aprendizado de máquina também caracteriza-se como fonte para pesquisa e inovação nesta área conforme apresentado discutidos por Ishakia [Ishakian et al. 2018] e Feng [Feng et al. 2018].

A computação *Serverless*, apesar de suas limitações, está cheia de desafios em Ciência da Computação para que pesquisadores possam investigar os problemas e propor soluções eficientes que possam contribuir com a ascensão de tecnologias *Serverless*; afirmação reforçada por Castro [Castro et al. 2019].

## 5.7. Prática

Nesta seção descrevemos os passos de instalação, criação de funções *Serverless*, suporte de uma nova linguagem, e executar exemplos de funções sobre a plataforma OpenFaaS. Nós escolhemos o OpenFaaS para a prática do minicurso devido às funcionalidades e características presentes nesta plataforma não serem encontradas em conjunto nas demais plataformas *Serverless* existentes e descritas neste minicurso.

### 5.7.1. Instalação

Para utilizar a plataforma pode-se escolher possuir uma instância da plataforma OpenFaaS instalada ou usar alguma plataforma disponível na nuvem.

#### 5.7.1.1. Instalação de uma instância da plataforma OpenFaaS

Para a instalação da plataforma OpenFaaS, é necessário ter instalado as aplicações:

- Git: `$ sudo apt-get install git`
- Docker: `$ sudo apt-get install docker.io`

Para instalar O OpenFaaS é necessário ter instalado o Git. Também clonar o repositório para o computador que irá hospedar a plataforma. No terminal, execute o comando:

```
$ git clone https://github.com/openfaas/faas.git
```

Serão baixados cerca de 14,5 MB de arquivos para o computador. Após o *download* execute os comandos:

```
$ sudo su
$ cd faas
$ make
```

Com isso a plataforma será instalada, diversas imagens de contêineres serão baixadas e construídas. Esse processo pode levar vários minutos. Ao final desse processo a plataforma ainda não estará ativa. Ainda é necessário construir e publicar. Para construir use o *script* build via o comando:

```
$ ./build.sh
```

Nesse ponto serão construídos todas as dependências que a plataforma necessita para seu completo funcionamento. Ao final do processo a plataforma está apta a ser publicada através do comando:

```
$ docker swarm init --advertise-addr [interface de rede]
$ ./deploy_stack.sh
```

Após executar os comandos anteriores serão criadas as credenciais de usuário e senha para acessar o serviço OpenFaaS. Guarde as credenciais, elas serão necessárias mais tarde. Se tudo estiver correto a plataforma já está apta a ser usada para o envio de funções. É possível, nesse momento, acessar a plataforma por meio de um navegador, digite na barra de endereços 127.0.0.1 : 8080.

No primeiro acesso será necessário se autenticar fornecendo as credenciais geradas quando foi feito a publicação do OpenFaas. A figura 5.5 mostra a janela de autenticação. Preencha os campos de usuário e senha clique em *OK*.

#### 5.7.1.2. Instalação da aplicação Cliente

Para utilizar os serviços de desenvolvedor da plataforma, é necessário ter instalado uma aplicação cliente do OpenFaaS, o *faas-cli*. Exitem três modos básicos de instalação via *script*, via *brew* e uma terceira forma para usuários do sistema operacional Windows. Aqui descreveremos a instalação via *script*. No terminal execute o comando:

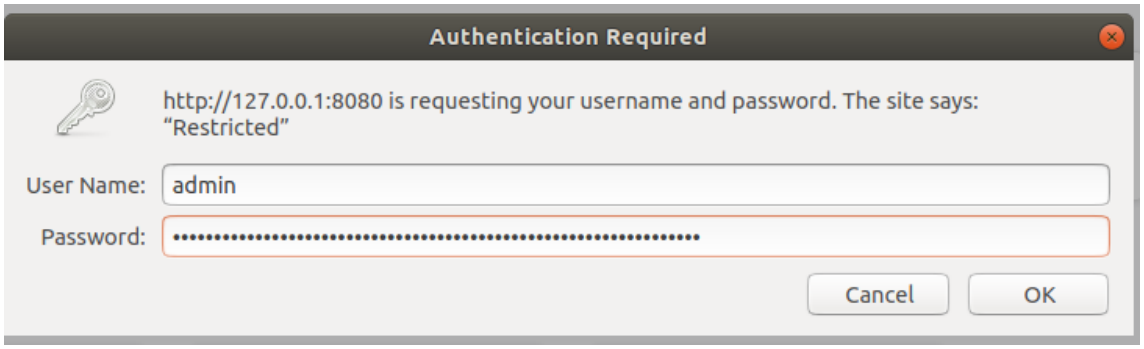



**Figura 5.5. Exemplo da janela de autenticação da plataforma via navegador.**

```
$ curl -sSL https://cli.openfaas.com | sudo sh
```

Caso não ocorra nenhum erro no processo de instalação a aplicação cliente do OpenFaas está pronta para divulgar funções em alguma plataforma OpenFaaS. o *gateway* por padrão é 127.0.0.1 : 8080. É realizar o login no *gateway* usando as credenciais obtidas no passo anterior:

```
faas-cli login [--username USERNAME] [--password PASSWORD]
```

Caso deseje fazer o *download* de todos os *templates* oficiais basta executar:

```
$ faas-cli template pull
$ ls template/
```

### 5.7.2. Implementação de funções

Com o *faas-cli* instalado para começar a escrever as funções basta, em um diretório arbitrário[5], digitar o seguinte comando:

```
$ faas-cli new --lang [linguagem] [nome da função]
```

Após esse comando serão criados alguns arquivos no diretório. Sem perda de generalidade é apresentado agora um exemplo de criação de função utilizando a linguagem Python. Execute o comando:

```
$ faas-cli new --lang python hello-minicurso
```

Serão criados 3 arquivos. Um no diretório raiz e dois em uma pasta com o nome que foi dado para a função

```
hello-minicurso/handler.py
hello-minicurso/requirements.txt
hello-minicurso.yml
```

[5]É recomendável um diretório exclusivo para isso.

A função de cada um dos arquivos será explicada posteriormente, ainda nessa seção. É necessário editar cada um dos três arquivos.

Primeiro edita-se o arquivo **handler.py**

**Código 1** Conteúdo do arquivo handler.py.

```
1  def handle(nome):
2      print("Ola: " + nome)
```

O arquivo **handler.py** com o código 1 é a função propriamente dita e, neste caso, simplesmente imprime na tela "Olá xxx", sendo xxx o que é fornecido à função como parâmetro.

No Arquivo **hello-minicurso.yml** são definidas as opções tal qual a plataforma a ser usada, qual o *gateway*, e quais as funções serão executadas. O código 2 mostra o conteúdo do arquivo.

**Código 2** Conteúdo do arquivo hello-minicurso.yml

```
1  provider:
2    name: faas
3    gateway: http://localhost:8080
4
5  functions:
6    hello-minicurso:
7      lang: python
8      handler: ./hello-minicurso
9      image: hello-minicurso
```

- *gateway* - Especifica o servidor no qual sua função irá rodar.
- *functions* - Este bloco define as funções. Há a possibilidade de mais de uma função por arquivo.
- *lang:* especifica em qual linguagem de programação a sua função foi escrita.
- *handler* especifica qual o caminho até sua função.
- *image* especifica qual o nome da imagem Docker.

No arquivo **requirements.txt**, específica qualquer dependência que seja necessária para executar a função. Neste exemple específico não será editado. Então pode-se construir a função com o comando:

```
$ faas-cli build -f ./hello-minicurso.yml
...

Successfully tagged hello-minicurso:latest
Image: hello-minicurso built
```

Com a imagem *Serverless* gerada pode-se publicar função:

```
$ faas-cli deploy -f ./hello-minicurso.yml
```

Caso a função seja publicada sem nenhum erro, há um retorno da plataforma com código 200 e o endereço para invocar a função:

```
200 OK
URL: http://localhost:8080/function/hello-minicurso
```

Neste ponto a função está publicada e pronta pra ser invocada.

#### 5.7.2.1. Invocar funções

O OpenFaaS permite que usuários se conectem à plataforma invocando funções *Serverless* a partir de softwares cliente. Cliente é o componente do software responsável por interagir com o usuário através de interfaces ou linhas de comando. No OpenFaaS, clientes são utilizados para manipular funções. Para que a comunicação entre o software cliente e a plataforma ocorra, é necessário que o cliente tenha suporte ao protocolo HTTP.

Ao invocar uma função, o usuário pode fornecer parâmetros, por exemplo, uma sequência finita de números inteiros para uma função que calcule a média entre os valores fornecidos. Os parâmetros e o endereço da função solicitada são armazenados em uma requisição HTTP e enviados para o *API Gateway*, que redireciona o conteúdo para o componente *watchdog* da função correspondente. Os dados são processados e o resultado é enviado de volta para o software cliente. A seguir, apresentamos três clientes compatíveis com o OpenFaaS.

**cURL (*client URL* ou URL)**: é uma ferramenta de linha de comando para transferência e recebimento de dados usando sintaxe URL. De acordo com o manual oficial [Stenberg 1997], cURL usa a biblioteca libcurl e suporta todos os protocolos que o libcurl suporta, como HTTP, FTP e LDAP. cURL pode ser utilizado em linha de comando ou integrado a *scripts* executados por outros programas como *Makefiles* ou *scripts* escritos em Python. cURL está disponível para a maioria das versões do Linux e pode ser instalado via gerenciador de pacotes da distribuição. No Ubuntu, o comando de instalação é:

```
$ sudo apt-get install curl
```

Para invocar uma função via cURL o usuário deve fornecer quatro parâmetros: (1) endereço IP do OpenFaaS, sendo o endereço padrão localhost ou 127.0.0.1; (2) porta do componente API Gateway, por padrão a porta 8080; (3) nome da função *Serverless* e (4) os parâmetros da função, se existirem. No último caso, a opção -d deve ser adicionada. A seguir, apresentamos a estrutura do comando cURL e invocamos a função *hello-minicurso*. O resultado desta função é a mensagem enviada pelo usuário como parâmetro via cURL.

```
$ curl <ip>:<porta>/function/<nome> -d <parâmetros>
$ curl localhost:8080/function/hello-minicurso -d "oi"
```

**FaaS CLI**: é a interface de linha de comando oficial do OpenFaaS. FaaS CLI possui vários comandos para gerenciar funções *Serverless*. A lista completa de opções pode ser obtida através do comando `$ faas-cli help`.

A opção que permite invocar funções é a `invoke`. Em comparação com cURL, `invoke` requer menos parâmetros, sendo necessário informar somente o nome da função desejada. A seguir, apresentamos um exemplo que invoca a função *hello-minicurso* com o comando `faas-cli`:

```
$ faas-cli invoke hello-minicurso
```

Ao executar o comando acima, uma mensagem de início de leitura será exibida no terminal. Após a mensagem, o FaaS CLI aguarda parâmetros de entrada que podem ser inseridos a qualquer momento pelo usuário. A função *Serverless* só será efetivamente invocada após o encerramento do processo de leitura dos parâmetros de entrada. A seguir, apresentamos um exemplo completo do processo de invocação de uma função com FaaS CLI. O parâmetro fornecido foi a frase bom dia"e o resultado obtido através da função *hello-minicurso* foi a mensagem "Ola: bom dia".

```
$ faas-cli invoke hello-minicurso
Reading from STDIN - hit (Control + D) to stop.
bom dia
Ola: bom dia
```

**UI Portal** ***(User Interface Portal)***: é o cliente web oficial do OpenFaaS. Ele apresenta uma lista de funções ativas e possui uma interface que permite inserir parâmetros e invocar funções. UI Portal pode ser acessado em qualquer navegador nos endereços `http://localhost:8080/` ou `http://127.0.0.1:8080/`. Quando uma função é selecionada, dois formulários (figuras 5.6 e 5.7) são exibidos simultaneamente no navegador. O primeiro formulário (figura 5.6) apresenta informações da função como status, total de chamadas realizadas, número de réplicas, imagem Docker e URL. Existe ainda um botão de exclusão que finaliza o processo de execução do contêiner e remove a função da lista de funções disponíveis.

O segundo formulário permite invocar a função selecionada. Os parâmetros de entrada são fornecidos através do campo de texto *request body*, ou corpo da requisição. O resultado é apresentado no campo *response body*, ou corpo da resposta. O usuário pode escolher vê-lo no formato de texto sem nenhuma formatação ou como JSON (*JavaScript Object Notation*, ou Notação de Objetos do JavaScript), um formato compacto de estruturação de dados como objetos. Existe ainda a opção de *download*, em que o conteúdo da resposta é armazenado em um arquivo que é baixado em seguida.

Além disso, informações referentes à execução da função, como o código de status da resposta e o tempo total de execução da função, são apresentados em campos próprios.

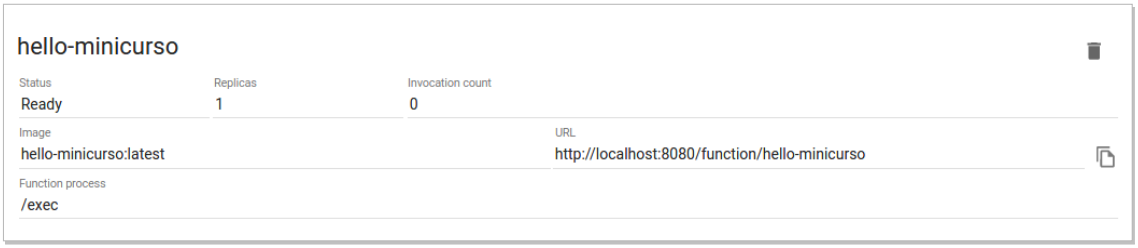



**Figura 5.6. Dados da função apresentados pelo UI Portal.**

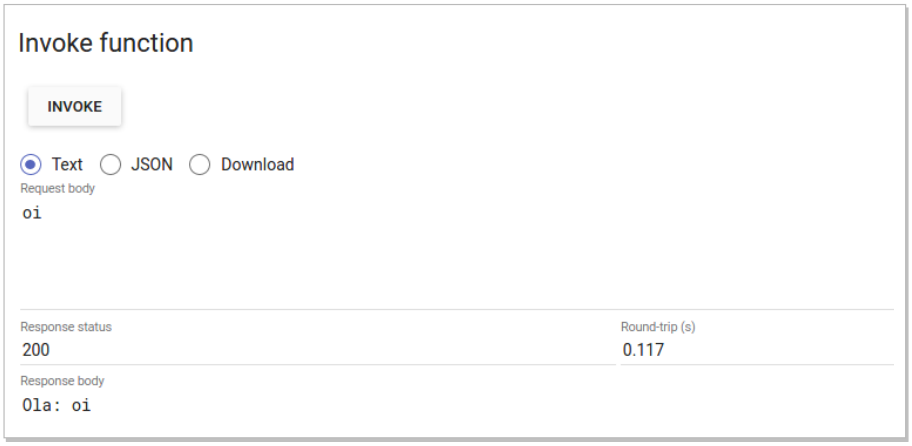



**Figura 5.7. Ferramenta de execução de testes do UI Portal.**

### 5.7.3. *Templates*

*Templates* são conjuntos de arquivos usados como modelo para criar novas funções compatíveis com OpenFaaS. *Templates* podem ser criados para suportar qualquer linguagem. OpenFaaS suporta onze *templates* oficiais que podem ser customizados de acordo com a necessidade do usuário. Um *template* é composto pelos seguintes arquivos:

- **Dockerfile:** Contém instruções específicas utilizadas pelo motor Docker para criar uma nova imagem;
- ***Index*:** Arquivo responsável por manipular a entrada e saída padrão. Este arquivo funciona como interface entre o componente *watchdog* e arquivo *handler* recebendo parâmetros de entrada, e retornando resultados de processamento;
- ***Handler*:** Arquivo que recebe parâmetros de entrada do *index*, inicializa o processamento, e retorna o resultado para interface;
- ***Template.yml***: Contém comandos de configuração de serviços disponíveis para função.

Os arquivos *index* e *handler* podem ser escritos na linguagem do *template*, e podem interagir com outros elementos do OpenFaaS, por exemplo, *Makefiles* e *scripts*. Algumas linguagens como Java, PHP e Python podem exigir um número maior de arquivos,

por exemplo, classes, *scripts* de configuração, e listas de dependências. Mas, de modo geral, todos os *templates* possuem a mesma estrutura básica apresentada nesta seção.

#### 5.7.3.1. Inserindo um *template* no OpenFaaS

OpenFaaS possui um repositório denominado *Classic Templates* [Ellis 2017e] exclusivo para *templates* oficiais. A ferramenta FaaS CLI descrita na seção 5.7.2.1 acessa o repositório durante a criação de uma nova função em busca de um *template* que corresponde a linguagem solicitada. O repositório é clonado automaticamente se não for encontrado no diretório em que o comando `faas-cli new` for executado. No minicurso, criamos um novo *template* para a linguagem C. No entanto, esse procedimento pode ser realizado para qualquer linguagem de programação desde que dependências como compiladores e arquivos do programa da função sejam alterados devidamente.

O primeiro passo para inserir um *template* é criar um novo diretório de trabalho ou acessar um diretório existente. Se o conjunto de *templates* oficiais ainda não estiver disponível no diretório, o comando `faas-cli template pull` pode ser utilizado para obtê-lo. A seguir, apresentamos a sequência inicial de comandos.

```
$ mkdir -p ~/functions && cd ~/functions
$ faas-cli template pull
$ cd template && ls
$ mkdir c-language && cd c-language
```

O último comando cria e acessa a pasta para o *template* a ser customizado. Os arquivos *header.h* e *handler.h* são criados logo após a execução dos comandos apresentados no primeiro passo. Esses arquivos são responsáveis por processar os parâmetros de entrada e produzir os resultados. É interessante mantê-los em um diretório exclusivo dentro da pasta do *template*. Nós criamos uma pasta chamada *function* para armazenar esses arquivos.

```
$ mkdir function/ && cd function/
$ touch header.h handler.h
```

O arquivo *header* inclui bibliotecas básicas e define o cabeçalho da função *handler*. No código 3 apresentamos o escopo da função *handler*. Ela recebe como entrada uma sequência de caracteres e retorna um valor inteiro.

**Código 3** Escopo da função *handler*.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int handler(char *entrada);
```

O código 4 representa o conteúdo da função *handler*. Dentro da função *handler* o usuário pode definir o comportamento da função *Serverless*, e também distribuir o processamento através de outras funções. Como exemplo ilustrativo, exibimos apenas uma mensagem concatenada com a *string* fornecida como parâmetro de entrada.

**Código 4** Função *handler*.

```c
#include "header.h"

int handler(char *entrada) {
    printf("Oi! A mensagem recebida foi '%s'\n", entrada);
    return 0;
}
```

O próximo arquivo a ser criado é o *index.c*. Ele deve ser criado ao lado da pasta *function/*, isto é, em *~/functions/template/language-c/*.

```
$ cd ..
$ touch index.c
```

O arquivo *index* é responsável por receber parâmetros a partir da entrada padrão e enviá-los ao *handler*. No codigo 5 o contéudo do arquivo *index* é apresentado. A variável `TAM_MAX_DADOS` define o tamanho máximo para a sequência de caracteres de entrada. O arquivo pode ser alterado para converter a entrada em outros tipos de dados.

**Código 5** Contéudo arquivo *index*.

```c
#include "header.h"

#define TAM_MAX_DADOS 16

int main() {
    char *dados;
    int tamanho;

    // Alocando a memória para o vetor de dados;
    dados = malloc(TAM_MAX_DADOS * sizeof(char));

    // Armazenando dados da entrada padrão;
    fgets(dados, TAM_MAX_DADOS, stdin);
    tamanho = strlen(dados);

    // Removendo '\n' do final do vetor;
    if (dados[tamanho - 1] == '\n')
       dados[--tamanho] = '\0';

    // Enviando dados recebidos para o handler;
    handler(dados);
    return 0;
}
```

Após todos os arquivos do *template* serem criados, o próximo passo é escrever os arquivos *Dockerfile* e *template.yml*, utilizados para configurar a imagem Docker e os serviços necessários para converter a função criada em um contêiner compatível com OpenFaaS. Os arquivos devem ser criados em *~/functions/template/language-c/*, ao lado da pasta *functions/* e do arquivo *index*.

```
$ touch Dockerfile template.yml
```

*Dockerfile* é um arquivo com sintaxe própria que descreve as etapas a serem executadas pelo Docker para criar uma nova imagem. Ele inclui comandos para instalar pacotes, criar arquivos e diretórios, executar arquivos, criar variáveis de ambiente, entre outros. A imagem gerada após a execução do *Dockerfile* é utilizada como modelo para a criação dos contêineres da função que serão implantados no OpenFaaS. No código 6 o conteúdo do arquivo é apresentado.

**Código 6** Contéudo arquivo *Dockerfile*.

```
1  FROM openfaas/classic-watchdog:0.18.0 as watchdog
2
3  FROM alpine:latest
4
5  RUN apk add build-base
6
7  COPY --from=watchdog /fwatchdog /usr/bin/fwatchdog
8  RUN chmod +x /usr/bin/fwatchdog
9
10 WORKDIR /app/
11
12 COPY function/header.h      .
13 COPY function/handler.c     .
14 COPY index.c                .
15
16 RUN gcc header.h handler.c index.c -static -o /exec \
17 && chmod +x /exec
18
19 ENV fprocess="/exec"
20
21 HEALTHCHECK --interval=3s CMD [ -e /tmp/.lock ] || exit 1
22
23 CMD ["fwatchdog"]
```

De modo geral, as tarefas realizadas por cada comando no *Dockerfile* são descritos na ordem em que aparecem no código 6. Os comandos presentes nas linhas 1, 3, 7, 8, 10, 19, 21 e 23 são comuns à maioria dos *templates*. Os comandos das linhas 5, 12, 13, 14, 16 e 17 podem variar de acordo com a linguagem. Além disso, é possível adicionar instruções para realizar *download* e instalação de dependências, execução de *scripts* e outros, conforme a necessidade do usuário. O repositório oficial de *templates* do OpenFaaS [Ellis 2017e] possui vários exemplos que podem ser utilizados como exemplo e material de estudo.

1. **Linha 1:** Importa a imagem do componente *watchdog*;

2. **Linha 3:** Importa a imagem do *Linux Alpine*. Alpine é utilizado como ambiente de execução devido ao tamanho mínimo de sua estrutura base (5 MB excluindo o *kernel* e outras dependências);

3. **Linha 5:** Instala o pacote *build-base* que contém o compilador GCC 5.3.0 e outros pacotes essenciais como *binutils* e *libc-dev*;

4. **Linhas 7 e 8:** Copiam o conteúdo do componente *watchdog* da imagem oficial para nova imagem e concedem permissão para execução;

5. **Linha 10:** Define o diretório */app/* como ambiente de trabalho do contêiner. Comandos como `COPY`, `RUN` e `CMD` passam a ser executados nesse diretório;

6. **Linhas 12, 13 e 14** Copiam os arquivos do *template* para o diretório de trabalho definido na linha 10;

7. **Linhas 16 e 17:** Compilam os arquivos obtidos com o compilador GCC e geram o executável do arquivo *index.c*;

8. **Linha 19:** Define o executável criado como processo principal do contêiner. Este comando é obrigatório. Sem ele, dados enviados ao contêiner não serão recebidos pelo arquivo *index*;

9. **Linha 21:** Define em 3 segundos o intervalo de atuação do mecanismo que impede que o contêiner em uso seja alocado por outro usuário;

10. **Linha 23:** Inicializa o *watchdog* que passará a receber solicitações da aplicação cliente via API *Gateway*.

O último arquivo a ser configurado é o *template.yml*. Este arquivo é responsável por informar aos serviços do OpenFaaS qual é o principal arquivo executável da função. Ele serve também para descrever rotinas de serviços específicos implementados pelo usuário. O conteúdo do arquivo *template.yml* é apresentado no código 7.

**Código 7** Contéudo arquivo *template.yml*.

```
language: c
fprocess: /exec

welcome_message:
    Parabéns! Você criou uma função serverless em C!
    Não se esqueça de atualizar o Dockerfile ao adicionar
    novos arquivos.
```

Após a sequência de passos realizados, todos os arquivos necessários para suportar uma nova linguagem foram configurados no OpenFaaS.

### 5.7.4. Criando funções eBPF no OpenFaaS

Nesta seção discutimos como adicionar suporte eBPF no OpenFaaS e apresentamos exemplos práticos. eBPF é uma tecnologia utilizada no processamento de pacotes que provê programabilidade no plano de dados. OpenFaaS pode ser utilizado como um sistema de processamento de pacotes *Serverless* totalmente programável e escalável usando funções *Serverless* eBPF.

#### 5.7.4.1. eBPF (*extended Berkeley Packet Filter*)

É uma máquina virtual com arquitetura RISC de 64 bits de propósito geral. Foi inserida no *kernel* do Linux a partir da versão 3.15. eBPF realiza processamento rápido de pacotes de forma independente de protocolos e em tempo de execução dentro do *kernel*, provendo programabilidade na computação de pacotes. Programas escritos em eBPF são compilados em *bytecode* eBPF antes de serem executados no *kernel*. Algumas linguagens como P4 e C já suportam essa tecnologia. O minicurso intitulado *Processamento Rápido de Pacotes com eBPF e XDP* [Vieira et al. 2019] e o tutorial *Fast packet processing with eBPF and XDP* [Vieira et al. 2020] apresentam mais informações sobre essa tecnologia.

#### 5.7.4.2. Adicionando suporte uBPF no *template* C

Nessa seção vamos utilizar o uBPF [Iovisor 2020] que nada mais é do que uma implementação da maquina virtual eBPF em espaço de usuário.

A configuração apresentada na seção 5.7.3.1 não é suficiente para compilar e executar códigos em C uBPF. Para isso, é necessário instalar pacotes de dependências e compilador Clang, além de gerar a maquina virtual. As instalação desses pacotes pode ser feito alterando os arquivos do *template* da linguagem C apresentado na seção 5.7.3.1. Outra alternativa pode ser copiar o conteúdo da pasta *c-language* para criar um *template* exclusivo para C uBPF, mantendo o *template* da linguagem C intacto como apresentado abaixo.

```
$ cd ~/functions/template
$ cp -r c-language c-ebpf && cd c-ebpf
```

Precisamos alterar nosso *Dockerfile* conforme pode ser visto no código 8. As linhas 5 a 8 instalam as dependências, as linhas 10 a 12 baixam o código do uBPF do repositório oficial e compilam a maquina virtual ubpf de teste, a linha 16 compila o código uBPF escrito pelo usuário no arquivo handle.c e, por fim a linha 26 executa o código gerado na maquina virtual. Diferente do *template c-language* do exemplo anterior, neste *template* usaremos apenas o arquivo handle.c que deve ser alterado conforme o código 9.

#### 5.7.4.3. Exemplo usando uma função no contêiner uBPF

Nesta seção apresentamos um exemplo trivial, usando o *template* gerado na seção 5.7.4.2.

**Código 8** Contéudo do arquivo *Dockerfile* com suporte uBPF.

```
FROM openfaas/classic-watchdog:0.18.0 as watchdog

FROM alpine:latest

RUN apk add build-base \
      clang \
          git \
          llvm
WORKDIR /home/app
RUN git clone https://github.com/iovisor/ubpf.git
WORKDIR ubpf
RUN make -C vm

COPY function/handler.c .

RUN  clang -O2 -emit-llvm -c handler.c -o - | \
llc -march=bpf -filetype=obj -o handler.o

# Add non root user
# Create a non-root user
RUN addgroup --system app \
    && adduser --system --ingroup app app

RUN chown app:app -R /home/app
USER app

ENV fprocess="/home/app/ubpf/vm/test /home/app/ubpf/handler.o"

# Set to true to see request in function logs
ENV write_debug="false"

EXPOSE 8080

HEALTHCHECK --interval=3s CMD [ -e /tmp/.lock ] || exit 1

CMD ["fwatchdog"]
```

**Código 9** Contéudo do arquivo *handler.c* com suporte uBPF.

```
#include "vm/inc/ubpf.h"

int  main() {

}
```

Primeiro instanciamos uma função baseada no *template*.

```
$ faas-cli new --lang c-ebpf hello_ebpf
$ cd hello_ebpf/function/
```

Adicionamos o código que queremos executar. Neste exemplo vamos apenas elevar uma constante ao quadrado, bastando para isso editar o arquivo handler.c conforme o código 10.

**Código 10** Conteúdo do arquivo *handler.c*.

```
1 #include "vm/inc/ubpf.h"
2
3 int  main() {
4     int a;
5     a=2;
6     return a*a;
7 }
```

Podemos gerar a imagem com o comando:

```
$ faas-cli build -f ./hello-ebpf.yml
```

Com a imagem *Serverless* gerada, pode-se publicar função:

```
$ faas-cli deploy -f ./hello-ebpf.yml
```

Em seguida, evocar a função e receber o resultado esperado.

```
$ curl http://127.0.0.1:8080/function/hello-ebpf
$ 4
```

Apesar de simples, o exemplo demonstra como, com essa ferramenta em mãos, se pode rapidamente gerar e disponibilizar funções ebpf.

## 5.8. Conclusões e Discussões Finais

Este minicurso apresentou o estado da arte em Computação *Serverless*. Foram discutidos os conceitos, as limitações, os casos de uso, as áreas de pesquisa em aberto e as plataformas disponíveis no mercado.

Exemplos práticos de implementação de funções *Serverless* utilizando as principais ferramentas de código aberto e algumas plataformas de provedores de serviços em nuvem foram demonstradas de modo a aprofundar o estudo do ecossistema da computação *Serverless*.

Discutiu-se também, o futuro desse emergente modelo de computação, que apesar de permitir implementações distribuídas, tem limitações claras relacionadas à latência na

troca de mensagens, persistência e compartilhamento de dados e a falta de conhecimento global de modo a permitir algoritmos mais eficientes.

Tudo isso abre oportunidades de pesquisa que remetem a tópicos e problemas clássicos em Ciência da Computação como otimizações em Sistemas Operacionais, Sistemas Distribuídos, Localidade de dados em Redes, Carregamento eficiente de programas e segurança.

Este minicurso contribui trazendo uma visão abrangente sobre a computação *Serverless* de modo que o leitor possa compreender o estado da arte e para onde caminha essa tecnologia. Além disso, contribui com um arcabouço rico em exemplos de implementação nas principais ferramentas de código aberto e do mercado facilitando assim o aprendizado e ambientação com essa emergente área de conhecimento e pesquisa.

## Agradecimentos

Agradecemos as agências de pesquisa CNPq, FAPESP projeto 2018/23085-5 e FAPEMIG pelo apoio financeiro. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

## Referências

[Alpine Linux 2018] Alpine Linux (2018). `https://alpinelinux.org/about/`. Acessado em: 01/03/2020.

[Ao et al. 2018] Ao, L., Izhikevich, L., Voelker, G. M., and Porter, G. (2018). Sprocket: A serverless video processing framework. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '18, page 263–274, New York, NY, USA. Association for Computing Machinery.

[Apache 2004] Apache (2004). Apache license. `https://www.apache.org/licenses/LICENSE-2.0`. Acessado em: 20/03/2020.

[AWS Lambda 2014] AWS Lambda (2014). Aws lambda. `https://aws.amazon.com/lambda/`. Accessado em: 29/09/2019.

[Azure Functions 2016] Azure Functions (2016). Azure functions. `https://azure.microsoft.com/en-us/services/functions/`. Accessado em: 29/09/2019.

[Bila et al. 2017] Bila, N., Dettori, P., Kanso, A., Watanabe, Y., and Youssef, A. (2017). Leveraging the serverless architecture for securing linux containers. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 401–404. IEEE.

[Castro et al. 2019] Castro, P., Ishakian, V., Muthusamy, V., and Slominski, A. (2019). The rise of serverless computing. *Commun. ACM*, 62(12):44–54.

[Choi et al. 2019] Choi, S., Shahbaz, M., Prabhakar, B., and Rosenblum, M. (2019). λ-nic: Interactive serverless compute on programmable smartnics. *arXiv preprint arXiv:1909.11958*.

[Church et al. 2020] Church, M., Ruiz, M., Seifert, A., and Marshall, T. (2020). Docker - docker swarm reference architecture: Exploring scalable, portable docker container networks.

[CNCF landscape 2020] CNCF landscape (2020). Cncf serverless landscape. `https://landscape.cncf.io/format=serverless`. Accessado em: 08/03/2020.

[CouchDB 2005] CouchDB (2005). Couchdb. `https://couchdb.apache.org/`. Acessado em: 07/03/2020.

[Docker Inc. 2008] Docker Inc. (2008). Docker. `https://www.docker.com/`. Acessado em: 07/03/2020.

[Docker Inc. 2014] Docker Inc. (2014). Swarm mode overview. `https://docs.docker.com/engine/swarm/`. Acessado em: 07/02/2020.

[Ellis 2016] Ellis, A. (2016). `https://github.com/openfaas/faas/`. Acessado em: 29/02/2020.

[Ellis 2017a] Ellis, A. (2017a). `https://github.com/openfaas/faas-netes/`. Acessado em: 01/03/2020.

[Ellis 2017b] Ellis, A. (2017b). `https://github.com/openfaas/faas-cloud/`. Acessado em: 01/03/2020.

[Ellis 2017c] Ellis, A. (2017c). `https://github.com/openfaas/nats-queue-worker`. Acessado em: 03/03/2020.

[Ellis 2017d] Ellis, A. (2017d). `https://github.com/openfaas-incubator/kafka-connector`. Acessado em: 03/03/2020.

[Ellis 2017e] Ellis, A. (2017e). `https://github.com/openfaas/templates/`. Acessado em: 17/02/2020.

[Ellis 2017f] Ellis, A. (2017f). Introducing functions as a service (openfaas). `https://blog.alexellis.io/introducing-functions-as-a-service/`. Acessado em: 01/03/2020.

[Ellis 2017g] Ellis, A. (2017g). Openfaas: Serverless functions made simple for docker and kubernates. `https://www.openfaas.com/`. Acessado em: 29/09/2019.

[Feng et al. 2018] Feng, L., Kudva, P., Da Silva, D., and Hu, J. (2018). Exploring serverless computing for neural network training. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 334–341. IEEE.

[Flexera 2019] Flexera (2019). Cloud computing trends: 2019 state of the cloud survey. https://www.flexera.com/blog/cloud/2019/02/cloud-computing-trends-2019-state-of-the-cloud-survey/. Acessado em: 19/11/2019.

[Fouladi et al. 2017] Fouladi, S., Wahby, R. S., Shacklett, B., Balasubramaniam, K. V., Zeng, W., Bhalerao, R., Sivaraman, A., Porter, G., and Winstein, K. (2017). Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 363–376.

[Glikson et al. 2017] Glikson, A., Nastic, S., and Dustdar, S. (2017). Deviceless edge computing: Extending serverless computing to the edge of the network. In *Proceedings of the 10th ACM International Systems and Storage Conference*, SYSTOR '17, New York, NY, USA. Association for Computing Machinery.

[Google Cloud Functions 2016] Google Cloud Functions (2016). Google cloud functions. `https://cloud.google.com/functions/`. Accessado em: 29/09/2019.

[Google Inc. 2015] Google Inc. (2015). Production-grade container orchestration. `https://kubernetes.io/`. Acessado em: 07/02/2020.

[Grafana Labs 2014] Grafana Labs (2014). The analytics platform for all your metrics. `https://grafana.com/grafana/`. Acessado em: 13/02/2020.

[Hellerstein et al. 2018] Hellerstein, J. M., Faleiro, J., Gonzalez, J. E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., and Wu, C. (2018). Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651*.

[Hendrickson et al. 2016] Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. (2016). Serverless computation with openlambda. In *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'16, pages 33–39, Berkeley, CA, USA. USENIX Association.

[Igor Sysoev 2005] Igor Sysoev (2005). Nginx. `https://www.nginx.com/`. Acessado em: 07/03/2020.

[Iovisor 2020] Iovisor (2020). iovisor/ubpf.

[Ishakian et al. 2018] Ishakian, V., Muthusamy, V., and Slominski, A. (2018). Serving deep learning models in a serverless platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 257–262. IEEE.

[Jonas et al. 2019] Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., Shankar, V., Menezes Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J., Popa, R. A., Stoica, I., and Patterson, D. A. (2019). Cloud programming simplified: A berkeley view on serverless computing. Technical Report UCB/EECS-2019-3, EECS Department, University of California, Berkeley.

[Kafka 2011] Kafka (2011). Kafka. `https://kafka.apache.org/`. Acessado em: 07/03/2020.

[Khan et al. 2019] Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., and Ahmed, A. (2019). Edge computing: A survey. *Future Generation Computer Systems*, 97:219 – 235.

[Linux Foundation 2016] Linux Foundation (2016). What is prometheus. https://prometheus.io/docs/introduction/overview/. Acessado em: 01/03/2020.

[Liu et al. 2019] Liu, M., Peter, S., Krishnamurthy, A., and Phothilimthana, P. M. (2019). E3: energy-efficient microservices on smartnic-accelerated servers. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 363–378.

[MIT 1980] MIT (1980). Mit license. https://en.wikipedia.org/wiki/MIT_License. Acessado em: 02/03/2020.

[OpenWhisk 2016] OpenWhisk, I. (2016). IBM OpenWhisk Project. http://developer.ibm.com/openwhisk/. Acessado em: 08/03/2020".

[Pacífico et al. 2020] Pacífico, R. D. G., Duarte, L. F. S., Nacif, J. A. M., and Vieira, M. A. M. (2020). Sistema de processamento de pacotes *Serverless*. In *XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos (SBRC)*, Rio de Janeiro, RJ, Brasil. SBC.

[Quevedo et al. 2019] Quevedo, S., Merchán, F., Rivadeneira, R., and Dominguez, F. X. (2019). Evaluating apache openwhisk-faas. In *2019 IEEE Fourth Ecuador Technical Chapters Meeting (ETCM)*, pages 1–5. IEEE.

[Shafiei et al. 2020] Shafiei, H., Khonsari, A., and Mousavi, P. (2020). Serverless computing: A survey of opportunities, challenges and applications.

[Shankar et al. 2018] Shankar, V., Krauth, K., Pu, Q., Jonas, E., Venkataraman, S., Stoica, I., Recht, B., and Ragan-Kelley, J. (2018). Numpywren: Serverless linear algebra. *arXiv preprint arXiv:1810.09679*.

[Stenberg 1997] Stenberg, D. (1997). Curl: The man page. https://curl.haxx.se/docs/manpage.html. Acessado em: 06/03/2020.

[Vieira et al. 2019] Vieira, M. A. M., Castanho, M. S., Pacífico, R. D. G., Santos, E. R. S., Câmara Júnior, E. P. M., and Vieira, L. F. M. (2019). Processamento Rápido de Pacotes com eBPF e XDP. In *Minicurso do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos (SBRC)*, Porto Alegre, RS, Brasil. SBC.

[Vieira et al. 2020] Vieira, M. A. M., Castanho, M. S., Pacífico, R. D. G., Santos, E. R. S., Júnior, E. P. M. C., and Vieira, L. F. M. (2020). Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications. *ACM Comput. Surv.*, 53(1).

[Vilalta et al. 2017] Vilalta, R., López, V., Giorgetti, A., Peng, S., Orsini, V., Velasco, L., Serral-Gracia, R., Morris, D., De Fina, S., Cugini, F., et al. (2017). Telcofog: A unified flexible fog and cloud computing architecture for 5g networks. *IEEE Communications Magazine*, 55(8):36–43.

[Wang et al. 2018] Wang, L., Li, M., Zhang, Y., Ristenpart, T., and Swift, M. (2018). Peeking behind the curtains of serverless platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 133–146, Boston, MA. USENIX Association.

[Werner et al. 2018] Werner, S., Kuhlenkamp, J., Klems, M., Müller, J., and Tai, S. (2018). Serverless big data processing using matrix multiplication as example. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 358–365. IEEE.

[Wu et al. 2018] Wu, D., Fang, B., Yin, J., Zhang, F., and Cui, X. (2018). Slbot: A serverless botnet based on service flux. In *2018 IEEE Third International Conference on Data Science in Cyberspace (DSC)*, pages 181–188. IEEE.

[Xiong et al. 2018] Xiong, Y., Sun, Y., Xing, L., and Huang, Y. (2018). Extend cloud to edge with kubeedge. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 373–377. IEEE.

[Zhang et al. 2019] Zhang, M., Zhu, Y., Zhang, C., and Liu, J. (2019). Video processing with serverless computing: A measurement study. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '19, page 61–66, New York, NY, USA. Association for Computing Machinery.

Organização:








Apoio


Realização:



Patrocínio:

Ouro





Prata

Bronze