UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

NILTON JOSÉ MOCELIN JÚNIOR

FLOWPRI: UM FRAMEWORK PARA GERENCIAMENTO DE LARGURA DE BANDA COM PRIORIDADE APLICADO EM UM CENÁRIO DE CIDADES INTELIGENTES

JOINVILLE
2022

NILTON JOSÉ MOCELIN JÚNIOR

FLOWPRI: UM FRAMEWORK PARA GERENCIAMENTO DE LARGURA DE BANDA COM PRIORIDADE APLICADO EM UM CENÁRIO DE CIDADES INTELIGENTES

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: PhD. Adriano Fiorese

JOINVILLE
2022

NILTON JOSÉ MOCELIN JÚNIOR

FLOWPRI: UM FRAMEWORK PARA GERENCIAMENTO DE LARGURA DE BANDA COM PRIORIDADE APLICADO EM UM CENÁRIO DE CIDADES INTELIGENTES

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: PhD. Adriano Fiorese

BANCA EXAMINADORA:

Dr. Adriano Fiorese (Orientador)
UDESC

Membros:

Dr. Charles Christian Miers
UDESC

Dra. Janine Kniess
UDESC

Joinville, 07 de dezembro de 2022

RESUMO

As cidades inteligentes são as cidades que empregam uma combinação de tecnologias de coleta, processamento e disseminação de dados para desenvolver soluções que permitem lidar com o seu crescimento de forma eficiente e sustentável. A quantidade de dispositivos conectados à Internet tem aumentado, sendo esperado que existam em 2025 próximo de 30.9 bilhões de dispositivos *Internet of Things* (IoT) conectados, enquanto que os não IoT devem chegar a 10 bilhões. As cidades inteligentes utilizam desses dispositivos para obter dados e enviar a um serviço de nuvem para processamento, de modo a fornecer aplicações e serviços para os cidadãos. Aplicações de cidades inteligentes podem necessitar de recursos de rede garantidos para funcionarem com a qualidade necessária. Além disso, os fluxos de dados das aplicações e das fontes de dados podem variar de importância conforme eventos ou contexto. Desta forma, sem o devido controle dos recursos de rede, os procedimentos envolvidos no funcionamento das cidades inteligentes ficam comprometidos. O principal problema que pode ser identificado é que, sem o devido tratamento, os dados de fluxos de serviços de cidade inteligente precisam competir por recursos na rede. No entanto, as redes convencionais utilizadas na Internet não são capazes de prover os dois elementos principais do tráfego de dados de cidades inteligentes, que são: *Quality of Service* (QoS) e priorização, de forma dinâmica. Desta forma, neste trabalho é assumido que a rede que conecta a cidade inteligente seja baseada na arquitetura *Software Defined Network* (SDN), de modo que permita o gerenciamento dinâmico dos componentes de rede por meio de estratégias desenvolvidas sobre o elemento central da arquitetura, o controlador SDN. Assim, é proposto o *framework FLOWPRI-SDN*, que é uma aplicação que executa sobre o controlador OpenFlow Ryu e fornece reserva de recursos de forma priorizada e distribuída para emissores que informam seus requisitos de tráfego por meio de um contrato e pode ser aplicado ao cenário de cidades inteligentes para suprir suas necessidades de QoS. Para isso, são utilizadas estratégias de divisão de recursos de rede em classes de serviços, com compartilhamento de largura de banda. Além disso, é proposto um mecanismo de comunicação entre controladores, para permitir que os controladores de diferentes domínios que tenham suporte ao *framework* possam trocar informações sobre os requisitos de QoS e priorização dos fluxos, permitindo que os domínios ao longo do caminho entre origem e destino de um fluxo, apliquem as mesmas políticas. Portanto, o objetivo deste trabalho é suprir as necessidades das aplicações de cidades inteligentes em termos de QoS e priorização, de modo que as estratégias aplicadas resultem em qualidade fim-a-fim em cenários mais otimistas. Além disso, se espera incentivar o desenvolvimento de novos tipos de aplicações para cidades inteligentes, levando em consideração os recursos de rede necessários.

Palavras-chave: Cidades Inteligentes, Redes Definidas por *Software*, Gerenciamento da Prioridade de Fluxos de Dados, Gerenciamento de Largura de Banda, Qualidade de Serviço.

ABSTRACT

Smart cities are cities that employ a combination of data collection, processing and dissemination technologies to develop solutions that allow them to deal with their growth in an efficient and sustainable way. The number of devices connected to the Internet has increased, and it is expected that in 2025 there will be close to 30.9 billion of connected IoT devices, while non-IoT devices should reach 10 billion. Smart cities use these devices to get data and send it to a cloud service for processing in order to provide applications and services to citizens. Smart city applications may need guaranteed resources to work with the required quality. Furthermore, data flows from applications and data sources can vary in importance depending on events or context. In this way, without proper control of network resources, the procedures involved in the operation of smart cities are compromised. The main problem that can be identified is that, without proper handling, data from smart city service flows have to compete for resources on the network. However, the conventional networks used on the Internet are not capable of dynamically providing the two main elements of smart city data traffic, which are: QoS and prioritization. In this way, in this work it is assumed that the network that connects the smart city is based on the SDN architecture, in a way that allows the dynamic management of the network components through strategies developed on the central element of the architecture, the controller SDN. Thus, the *framework FLOWPRI-SDN* is proposed, which is an application that runs on SDN Ryu controller and provides resource reservation in a prioritized and distributed way for senders that inform their traffic requirements through a contract and can be applied to the smart city scenario to fulfill your QoS needs. For this, resource division strategies are used in service classes, with bandwidth sharing. Also, a communication mechanism between controllers is proposed, to allow controllers from different domains that support *framework* to exchange information about QoS requirements and prioritization of flows, allowing domains to along the path between the source and destination of a flow, the same policies apply. Therefore, the objective of this work is to meet the needs of smart city applications in terms of QoS and prioritization, so that the applied strategies result in a quality close to that achieved with end-to-end. In addition, it is expected to encourage the development of new types of applications for smart cities, taking into account the necessary network resources.

Keywords: *Smart City*, *Software Defined Networking*, *Data Flow Priority Management*, *Bandwidth Management*, *Quality of Service*.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API *Application Programming Interface*

AS *Autonomous System*

BAM *Bandwidth Allocation Method*

BAMSDN *Bandwidth Allocation Model through Software Defined Networking*

CoAP *Constrained Application Protocol*

DHCP *Dynamic Host Configuration Protocol*

DiffServ *Differentiated Services*

DSCP *Differentiated Services Code Point*

FIFO *Fist In First Out*

FTP *File Transfer Protocol*

G-BAM *Generalized Bandwidth Allocation Model*

GPS *Global Positioning System*

HTB *Hierarchical Token Bucket*

HTL *High-To-Low*

HTML *HyperText Markup Language*

HTTP *Hyper Text Protocol*

ICMP *Internet Control Message Protocol*

IEEE *Institute of Electrical and Electronics Engineers*

IoT *Internet of Things*

IP *Internet Protocol*

IPv4 *Internet Protocol Version 4*

IPv6 *Internet Protocol Version 6*

ISP *Internet service provider*

KVM *Kernel-based Virtual Machine*

LR-WPAN *Low-Rate Wireless Personal Area Network*

LSP *Language Service Provider*

LTH *Low-To-High*

M2M *Machine-to-Machine*

MPLS *Multiprotocol Label Switching*

MQTT *Message Queue Telemetry Transport*

NFC *Near Field Communication*

OVS *Open vSwitch*

OVSDB *Open Virtual Switch Data Base*

qdisc *Queueing Disciplines*

QCI QoS *Class Identifier*

QoS *Quality of Service*

REST *Representational State Transfer*

RFC *Request for Comments*

RFID *Radio Frequency Identification*

SDN *Software Defined Network*

SLA *Service Level Agreenment*

TC *Traffic Control*

TCP *Transmission Control Protocol*

TCP/IP *The Internet Protocol Suite*

TOS *Type of Service*

TS *Traffic Shaping*

UDP *User Datagram Protocol*

VLAN *Virtual Local Area Network*

WLAN *Wireless Local Area Network*

SUMÁRIO

1 INTRODUÇÃO

Uma das maiores dificuldades das redes de comunicação em fornecer QoS é lidar com a natureza dinâmica dos requisitos dos fluxos de dados das aplicações que trafegam entre dispositivos (GULERIA, 2016). Os requisitos de QoS dos fluxos de dados das aplicações podem variar ao longo do tempo, devido ao ambiente e ao cliente ao qual servem. Neste sentido, cidades inteligentes são exemplos de cenários onde QoS e prioridade de fluxos tem grande importância.

As cidades inteligentes (*Smart City*) podem ser definidas como cidades que empregam tecnologias de coleta, processamento e disseminação de dados em conjunto com tecnologias de rede e computação, de modo a promover o desenvolvimento de aplicações que melhorem a qualidade de vida para seus cidadãos (GHARAIBEH et al., 2017). A base do conceito de cidades inteligentes é o desenvolvimento e implementação de tecnologias de *Big Data*, IoT e comunicação sem fio entre dispositivos (MYEONG; JUNG; LEE, 2018). No entanto, a maioria das arquiteturas consideram que a coleta de dados e sua transmissão são os pontos que merecem mais atenção no desenvolvimento de uma cidade inteligente (YIN et al., 2015).

Geralmente, as cidades possuem dispositivos eletrônicos condensados em diversas regiões como casas, prédios, ruas e hospitais (COSTA; OLIVEIRA, 2020). Dentre os dispositivos que podem ser utilizados estão os sensores químicos, sensores biológicos, dispositivos gerenciadores de energia elétrica, *gadgets*, *smartphones*, computadores, câmeras e dispositivos de áudio. A etapa de geração de dados de uma cidade inteligente consiste em utilizar esses dispositivos, principalmente os dispositivos IoT, para coletar dados de diversas origens, como dados de tráfego de veículos, poluição e do clima, e realizar ações sobre esses ambientes (WENGE et al., 2014).

Desta forma, esses dados devem ser requisitados ou enviados para processamento em plataformas de nuvem, permitindo armazenamento e acesso para aplicações *smart city*. Alguns dispositivos são capazes de se conectar diretamente com as plataformas e outros exigem um *gateway* com mais recursos de computação e comunicação para permitir seu funcionamento e trocar dados com redes que utilizam os protocolos da Internet (*Internet Protocol* (IP)) (TSCHOFENIG et al., 2015).

A natureza e características desses dados habilitam o desenvolvimento de diversas aplicações para cidades inteligentes. O trabalho de (YIN et al., 2015) categorizou as possíveis aplicações relacionadas à: serviços fornecidos diretamente pelo governo, aplicações que melhoram o dia a dia dos cidadãos, aplicações relacionadas aos negócios e empreendimentos, e por fim, relacionadas aos ambientes e construções. Cada categoria possui outros subdomínios próprios que compreendem os diversos interesses de uma cidade.

Nesse cenário, aplicações trocam dados com dispositivos podem necessitar de recursos garantidos para exercerem sua funcionalidade com a qualidade necessária. Além disso, os fluxos de dados dessas aplicações também variam de importância conforme eventos ou contexto (COSTA et al., 2015). Isso ocorre, por exemplo nas cidades inteligentes que sofrem com eventos como desastres naturais e acidentes, que dinamicamente exigem que determinados fluxos de

dados devam trafegar com mais prioridade, devido a importância que os dados possuem para o evento específico.

Sem o devido controle, fluxos de aplicações voltadas para cidades inteligentes disputam recursos de rede entre si e com fluxos de dados *background* diversos, de modo a não ter seus requisitos de QoS e priorização atendidos (NDIAYE; HANCKE; ABU-MAHFOUZ, 2017). No entanto, apesar da importância dessas aplicações para a cidade, não existem mecanismos que garantam a transmissão eficiente de dados entre cidade e plataforma. Esses fluxos de dados necessitam ser tratados com QoS de modo que os recursos de rede sejam bem utilizados e que os serviços da cidade não sejam prejudicados.

Por meio dos estudos das características de tráfego de (ITU, 2001b; KRUGER, 2017; AUTHORITY, 2018; CHEN; FARLEY; YE, 2004; MAINUDDIN; DUAN; DONG, 2021; MOCNEJ et al., 2018) é possível observar os requisitos necessários para os tipos de aplicações mais comuns que podem ser implementados nas cidades inteligentes, como tráfego de vídeo, áudio, arquivos e entre outros. Além disso, alguns desses trabalhos também exploram as características dos fluxos de dados IoT, como periodicidade de comunicação, largura de banda necessária e atraso tolerável.

Essas informações permitem levantar os perfis dos fluxos de dados que trafegam na cidade, permitindo criar formas de identificação de requisitos desses fluxos e implementar estratégias para o fornecimento dos recursos necessários, principalmente quanto de largura de banda é possível reservar/alocar de forma garantida para tais aplicações. Porém, no estado de rede atual, nada se pode fazer para que esses requisitos estejam disponíveis para os fluxos.

Nesse sentido, os equipamentos de rede atuais (legado), não foram feitos para suportar o gerenciamento da demanda de recursos e a programabilidade que as arquiteturas de cidades inteligentes necessitam (Zhang et al., 2018). Tais equipamentos fazem com que as redes baseadas na arquitetura da Internet tradicional sejam muito rígidas e se baseiem em princípios fixos (ALAM et al., 2020). Os componentes da estrutura não possuem suporte à programação, fornecendo apenas o serviço de entrega de pacotes com melhor esforço. Desta forma, a qualidade de serviço pode ficar comprometida, uma vez que os serviços não podem ser fornecidos de forma eficiente pela rede (Hu et al., 2020).

Para implementar as estratégias de gerenciamento, a arquitetura de redes definidas por *software* (SDN) juntamente com o protocolo OpenFlow, fornecem uma abordagem diferente da tradicional para garantir serviços com qualidade, quando separa o plano de controle do plano de dados (Zhang et al., 2018). Neste modelo, o uso dos recursos de rede disponíveis é gerenciado pelo plano de controle, composto por um elemento chamado controlador SDN.

Utilizando o controlador SDN, é possível implementar aplicações de gerenciamento que programem dinamicamente a rede. Este elemento é capaz de se comunicar com os dispositivos do plano de dados (*switches*) usando o protocolo OpenFlow e por meio de troca de mensagens, permite configurar ações que um *switch* deve tomar ao receber pacotes de um fluxo de dados.

Desta forma, visando suprir a necessidade que as aplicações para cidades inteligentes

possuem quanto aos requisitos de priorização e QoS dos seus fluxos de dados, é proposto a implementação de um *framework* de gerenciamento de recursos de rede para redes SDN de forma distribuída, chamado *FLOWPRI-SDN*. O *FLOWPRI-SDN* consiste de uma aplicação implementada sobre o controlador SDN Ryu, que atua em uma rede que tenha suporte a arquitetura SDN. Com este componente, é proposto separar o tipo de tráfego em categorias e utilizar estratégias de compartilhamento de largura de banda e *traffic shaping* para fornecer QoS e priorização, de acordo com o especificado pelos emissores dos fluxos de dados.

Nesse sentido, o objetivo deste trabalho é implementar um *framework* para priorização de tráfego inter-domínio, com garantia de uso de largura de banda. Tal objetivo visa melhor utilizar os recursos de rede SDN e em relação às cidades inteligentes, suprir as necessidades de gerenciamento e reserva de recursos que as mesmas possuem, priorizando os dados de suas aplicações em relação a um tráfego *background* que não faz parte dessas aplicações. Portanto, com o *framework*, deve ser possível tratar dinamicamente da prioridade dos fluxos de dados, bem como fornecer largura de banda de acordo com os requisitos das aplicações que necessitem.

1.1 ESTRUTURA DO TRABALHO

Este trabalho se caracteriza como uma pesquisa aplicada e como uma prova de conceito, uma vez que busca implementar conhecimentos específicos para solucionar um problema. Primeiramente, o Capítulo 1 é apresenta uma introdução acerca dos conceitos e problemas relacionados aos ambientes de cidades inteligentes, disponibilizando o contexto e introdução ao problema a ser resolvido e a solução proposta por meio do objetivo do trabalho. No Capítulo 2 são revisadas as arquiteturas de cidades inteligentes para entender o funcionamento e comportamento do cenário, bem como são apresentados os componentes de QoS que envolvem os processos de gerenciamento de recursos de rede e o funcionamento das redes baseadas na arquitetura SDN. No Capítulo 3, os trabalhos relacionados com a proposta desenvolvida são agrupados e comparados em termos de suas funcionalidades. O *framework FLOWPRI-SDN* é apresentado no Capítulo 4, onde são apresentados os componentes internos que o compõe. No Capítulo 5, são apresentados os testes de *overhead* e de aplicação do *FLOWPRI-SDN*, que buscam demonstrar o funcionamento da solução proposta. Por fim, o Capítulo 6 apresenta as considerações finais e conclui o trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capitulo, apresentamos a fundamentação teórica necessária para a compreensão geral do trabalho. O estado da arte das tecnologias no momento da escrita do trabalho auxilia na contextualização do cenário em que a proposta de gerenciamento de recursos foi definida. Desta forma, alguns tópicos relacionados à estruturação do trabalho são revisados.

2.1 CIDADES INTELIGENTES

O desenvolvimento urbano e tecnológico dos últimos anos contribuiu com o aumento populacional das cidades. Estima-se que 50% da população global vive nas cidades nos dias de hoje, mas que seguindo o crescimento, deve ultrapassar 70% em 2050 (ALLIANCE, 2015). Com isso, as cidades se tornarão cada vez maiores, ao mesmo tempo que problemas como poluição, congestionamento, baixa qualidade de vida e falta de recursos também tendem a crescer.

O interesse global tem se voltado a abordagens que permitam o desenvolvimento sustentável e organizado das cidades, investindo em estratégias para torná-las cada vez mais inteligentes. Entretanto, atualmente não existe uma definição completamente aceita como padrão para o termo **cidade inteligente** (OKAI; FENG; SANT, 2018).

Assim, uma definição relacionada a funcionalidade das cidades inteligentes sustenta que as cidades inteligentes devem oferecer soluções que melhorem a qualidade de vida das comunidades urbanas de uma maneira sustentável e justa (RAMÍREZ-MORENO et al., 2021). Além disso, essas cidades devem ser mais eficientes, a ponto de gerenciar melhor seus recursos, serviços e tecnologias, centrando o planejamento nas pessoas por meio de uma implantação inteligente de infraestrutura tecnológica, conforme Figura 1.

Figura 1 – *Smart City*.

Fonte: (ZHANG et al., 2017).

Ainda, segundo (GHARAIBEH et al., 2017), uma cidade inteligente emprega uma combinação de tecnologias de coleta, processamento e disseminação de dados em conjunto com tecnologias de redes e computação, com medidas de segurança dos dados e privacidade, que

incentivam a inovação de aplicativos para promover a qualidade geral de vida dos cidadãos e também cobrir as necessidades dos serviços de transporte, saúde, serviços públicos, entretenimento e serviços governamentais.

Na definição de (CISCO, c2022), uma cidade inteligente utiliza tecnologias digitais para conectar, proteger e melhorar a vida dos cidadãos. A cidade utiliza sensores IoT, câmeras de video, mídias sociais e outros meios, para agir como entrada de dados, dando constante *feedback* ao operador e aos cidadãos. Com isso, se pode fornecer serviços e aplicações que aproveitem os dados colhidos e melhorem a vida na cidade.

Alguns benefícios que as cidades inteligentes agregam são levantados em (CISCO, c2022), fazendo uma divisão de como as partes envolvidas se beneficiam:

- Para os administradores das cidades: Melhora o engajamento entre os cidadãos e otimiza operações por meio de inteligência de dados em tempo real e colaboração entre organizações.

- Para os cidadãos: Melhora a vida diária através dos serviços fornecidos. Cidades inteligentes oferecem visibilidade dos dados da cidade em tempo real para melhorar a mobilidade, a conectividade e os serviços de segurança.

- Para os negócios e organizações: Insere novas possibilidades, impulsiona novos fluxos de receita e desenvolvimento econômico. Torna possível melhorar o conhecimento sobre as atividades dos clientes e seus comportamentos.

- Para desenvolvedores e fornecedores de soluções inteligentes: O desenvolvimento de serviços e produtos é o combustível que fornece dados para a cidade. Isso permite que a cidade inteligente melhore a eficiência operacional de determinados serviços, o engajamento entre cidadãos e aumente a viabilidade econômica.

2.1.1 Arquiteturas para Cidades Inteligentes

Devido a se ter diferentes definições de *Smart City*, diversos autores propuseram arquiteturas focando em aspectos específicos. Essas arquiteturas definem como os desenvolvedores de aplicações para cidades inteligentes devem seguir para lidar com os sistemas heterogêneos, os dados criticos e os usuários finais (SANTANA et al., 2017).

O trabalho (ZUBIZARRETA; SERAVALLI; ARRIZABALAGA, 2016) propôs uma arquitetura *Smart City* baseada na coleta e tratamento de dados IoT, que se organiza em cinco camadas, sendo elas: camada de aplicação, camada de rede, camada de enlace, camada de serviço e camada de sensoriamento.

A camada de aplicação é responsável pelos serviços fornecidos para os clientes. Nesta camada ocorre uma alta densidade de tráfego, principalmente pelo protocolo *Hyper Text Protocol* (HTTP), utilizado pelos servidores de páginas *HyperText Markup Language* (HTML) na Internet. Segundo o autor, o sistema implementado para cidades inteligentes deve fornecer os serviços

de monitoramento da qualidade do ar, monitoramento de barulho, monitoramento e assistência de saúde, gerenciamento de desperdícios, transporte público inteligente, controle de tráfego, gerenciamento energético, luzes inteligentes e automação industrial.

Na camada de rede, ocorre a transmissão e compartilhamento de dados pela rede de sensores. Além disso, esta camada age como uma interface entre sistemas, fazendo com que redes heterogêneas possam trocar informações entre si. Portanto, diversas tecnologias de conexão devem ser suportadas como Wi-Fi e outras específicas para dispositivos móveis e IoT (4G, 5G e ZigBee, por exemplo).

A camada de enlace é responsável por transferir os dados vindos das camadas superiores pela região geográfica, ou seja, é responsável pela infraestrutura de rede. Ele divide a camada em dois grupos, sendo o primeiro de tecnologias baseadas em redes convencionais como as redes LAN, MAN e WAN, as quais fornecem maior taxa de transferência e menor latência. Já a segunda se refere às redes voltadas para dispositivos com restrição de energia e recursos (redes restritas), incluindo *Institute of Electrical and Electronics Engineers* (IEEE) 802.11, Bluetooth, *Near Field Communication* (NFC) e *Radio Frequency Identification* (RFID).

A camada de serviço é uma camada que fornece integração entre os serviços IoT. Esta camada é voltada para as aplicações, portanto, deve ser customizada de acordo com as necessidades específicas. Desta forma, os designers precisam fornecer protocolos e *Application Programming Interface* (API)s compatíveis para os desenvolvedores das aplicações IoT.

Por fim, a camada de sensoriamento, consiste de *clusters* de dispositivos equipados com sensores e atuadores que são utilizados para coleta de dados. A interconexão desses dispositivos facilita o transporte dos dados até um *gateway* por meio de protocolos específicos.

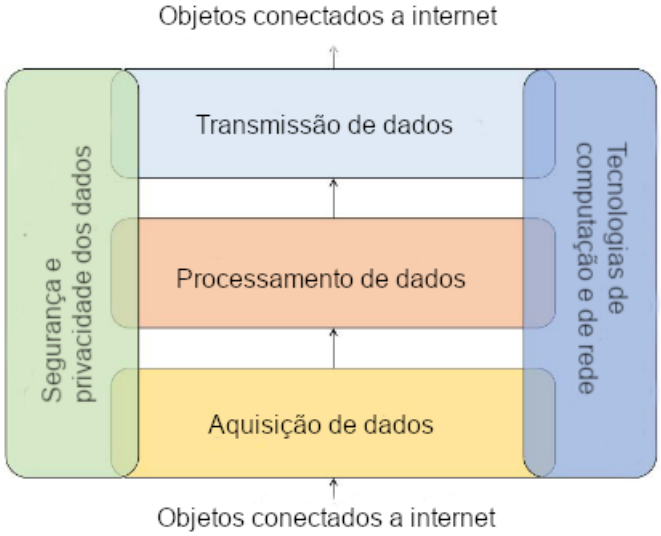
Em (SYED et al., 2021), o autor também propõe uma arquitetura *smart city* em cinco camadas. A camada de sensoriamento consiste de sensores e atuadores que coletam informações físicas para aplicações, bem como realizar ações sobre objetos físicos, como realizar a leitura de *tags* RFID. Os dados lidos na camada de sensoriamento são enviados para a camada de *Middleware* através da camada de rede, por meio de tecnologias de rede sem fio, como Wi-Fi e rede celular. A camada *Middleware* realiza o interfaceamento entre os dispositivos das camadas de sensoriamento e de aplicações, por meio de APIs e serviços de bancos de dados. Por último, é proposta uma camada de negócios, onde são implementados aplicativos para desenvolver estratégias para melhorar e gerenciar o sistema.

Ainda, considerando uma abordagem centrada no gerenciamento de dados, o trabalho (YIN et al., 2015) propôs uma arquitetura com estrutura hierárquica de quatro camadas: camada de aquisição de dados, camada de vitalização (tratamento) de dados, camada de dados comuns e serviços e a camada de aplicação. O objetivo desta estrutura é aproveitar a característica multidisciplinar, bem como as várias tecnologias de comunicação e suas aplicações, de uma cidade inteligente, na construção de modelos para análise e processamento de dados.

Diversas outras arquiteturas estão disponíveis na literatura, mas pode-se perceber alguns aspectos semelhantes entre elas, principalmente nas funções das camadas. Com isso, é possível

identificar, do ponto de vista dos dados, basicamente três etapas em todas as arquiteturas, a etapa de aquisição de dados na borda, a etapa de processamento dos dados em aplicações, e a etapa de disseminação, que transmite os dados processados para os usuários finais (Figura 2), como ocorre em (SANTANA et al., 2017; SINAEEPOURFARD et al., 2016; ESPOSTE et al., 2019; ZHANG, 2020; OGRODOWCZYK; BELTER; LECLERC, 2016).

Figura 2 – Ciclo de vida dos dados em uma *Smart City*.



Fonte: (GHARAIBEH et al., 2017).

Baseado nas descrições das arquiteturas de cidades inteligentes é possível observar os processos de comunicação envolvidos entre as camadas. Desta forma, sem o devido controle dos recursos de rede, os procedimentos envolvidos no funcionamento das cidades inteligentes fica comprometido. O principal problema que pode ser identificado é que os dados de fluxos de serviços de cidade inteligente precisam competir por recursos na rede contra fluxos genéricos.

2.1.2 Aquisição de Dados

O emprego de tecnologias de informação e o gerenciamento de dados em uma cidade inteligente são fundamentais para sua implementação (WENGE et al., 2014). As aplicações para cidades inteligentes dependem desses dados para fornecer soluções para a cidade. Assim, o primeiro desafio está em como obter os dados de uma cidade inteligente, que são gerados nas atividades cotidianas em grandes quantidades.

O trabalho (SINAEEPOURFARD; KROGSTIE; PETERSEN, 2018) classifica os dados em questão de tempo, como sendo: de tempo real, histórico e dados mais recentes. Os dados de tempo real são consumidos por aplicações que possuem restrição de distância e atraso. Já os dados históricos, são aqueles que devem ser acumulados e salvos para serem utilizados ou analisados posteriormente, por isso, são mais tolerantes quanto a distâncias e a latência. Por último, os dados *last-recent* (mais recentes) são os dados armazenados dos quais apenas os mais recentes são utilizados (SINAEEPOURFARD; KROGSTIE; PETERSEN, 2018).

Os dispositivos do tipo sensores possuem o papel mais importante na transição de uma cidade para cidade inteligente, uma vez que podem ser facilmente distribuídos em qualquer ambiente e permitem otimizar os recursos e serviços das cidades por meio da coleta de informações da cidade, cidadãos e das redes de comunicação em tempo real (RAMÍREZ-MORENO et al., 2021). Geralmente, as cidades possuem dispositivos eletrônicos condensados em diversas regiões como casas, prédios, ruas e hospitais (COSTA; OLIVEIRA, 2020). Desta forma, energia e conexão de rede geralmente não representam grandes problemas para esses dispositivos. Além disso, nesse cenário é possível implementar *gateways* que conectem os dispositivos às redes IP e agrupem os dados de redes de sensores em pacotes maiores, tornando essa abordagem genérica e válida para qualquer rede de dispositivos (COSTA; OLIVEIRA, 2020).

Alharbib e Soh (2019) identificam que os principais sensores utilizados em cidades inteligentes são de quatro tipos: biossensores, sensores eletrônicos, sensores químicos e *smart grid*. Os sensores eletrônicos possuem como principal função, detectar diferentes formas de energia por meio do uso de eletroscópio, anomalias magnéticas, detectores de tensão e entre outros (ALHARBI; SOH, 2019). Alguns exemplos de sensores dessa natureza são os sensores de vigilância de ambientes, sensores de estacionamento e sensores de velocidade.

Os sensores RFID utilizam de uma tecnologia que permite fornecer identificação e rastreamento sem fio a objetos. Nestes objetos, os dados podem ser armazenados em um dispositivo eletrônico chamado transpônder, que podem ser lidos ou gravados utilizando ondas de rádio (FINKENZELLER, 2010).

Uma *smart grid* é uma rede de sensores para gerenciar a rede elétrica. Esses sensores permitem medir informações sobre o comportamento dos fornecedores e consumidores de energia para que se possa atuar sobre a distribuição energética (ALHARBI; SOH, 2019).

Além desses citados, têm-se os biossensores, que são utilizados especificamente para a biomedicina e possuem a função de detectar e interagir com amostras biológicas, por exemplo anticorpos, enzimas e proteínas (UFSC, 2022). Sensores fotoelétricos utilizam de estímulos luminosos para gerar sinais de saída (COMERCIAL, 2020). Sensores térmicos reagem a estímulos térmicos de variação de temperatura para gerar sinais de saída (COMERCIAL, 2020).

Sensores químicos são sensores que medem elementos como dióxido de carbono e oxigênio (ALHARBI; SOH, 2019). Estes tipos de sensores conseguem identificar diversos problemas relacionados aos elementos químicos. Além disso, possuem grande área de atuação, por exemplo na área da saúde e em ambientes críticos.

Sistemas complexos de *Global Positioning System* (GPS) estão presentes em diversos dispositivos, como os presentes em veículos (PETROLO; LOSCRI; MITTON, 2017). Esta tecnologia permite obter informações de posicionamento que permitem fazer inferências, como sobre as condições do trânsito e descobertas de acidente.

Gadgets inteligentes também podem ser utilizados para obtenção de dados, por aplicações de cidades inteligentes, uma vez que são compostos de diversos sensores. *Smartphones* possuem acelerômetro, bússola, giroscópio, sensor de proximidade, sensor de luz, GPS, microfone, câmera

(USTEV; INCEL; ERSOY, 2013), etc.. Além disso, esses dispositivos geralmente possuem suporte a interfaces de rede sem fio, como nos *smartphones* que possuem suporte a Wi-Fi, 3G/4G/5G, *Bluetooth* e NFC (USTEV; INCEL; ERSOY, 2013).

Câmeras de vigilância são posicionadas em lugares como ruas e áreas movimentadas (SANTOS et al., 2021). Os dados coletados podem ser usados em algoritmos de reconhecimento facial para identificar pessoas e melhorar a segurança.

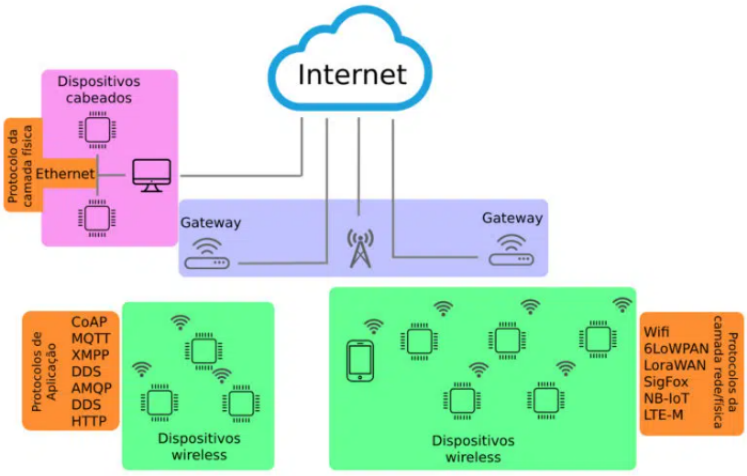
De outra perspectiva, se pode também considerar os cidadãos como sensores, quando podem relatar observações utilizando aplicativos específicos. O trabalho (BERNTZEN; JOHANNESSEN; FLOREA, 2016) cita o exemplo do *FixMyStreet.com* que permite que pessoas identifiquem problemas nas estradas do Reino Unido e informem o lugar e o problema utilizando uma interface Web.

2.1.3 Infraestrutura de Rede

Os dispositivos utilizados na coleta de dados são bastante heterogêneos, principalmente em questão de desempenho e conectividade. Anteriormente, os dispositivos eram conectados utilizando redes proprietárias, mas atualmente o foco tem se voltado em utilizar o conjunto de protocolos da Internet para comunicação entre os aplicativos e habilitar serviços avançados de comunicação (TSCHOFENIG; ARKKO, 2012).

Os dispositivos IoT podem ser classificados em duas categorias: restritos em recursos e ricos em recursos (AL-FUQAHA et al., 2015). Os dispositivos ricos em recursos são aqueles que possuem capacidades de suportar a pilha de protocolos *The Internet Protocol Suite* (TCP/IP) (ver Figura 3). Por outro lado, dispositivos que não possuem recursos para suportar TCP/IP não podem interoperar facilmente com os dispositivos que suportam, a não ser que utilizem de dispositivos intermediários que os habilitem a interagir com a pilha de protocolos TCP/IP.

Figura 3 – Protocolos IoT



Fonte: (PALARMINI, 2020).

Nestes casos, são utilizados outros protocolos e extensões de protocolos de rede que ajudam a otimizar as comunicações, principalmente em termos energéticos (TSCHOFENIG; ARKKO, 2012). A comunicação sem fio pode significar uma grande parte do consumo de energia para os dispositivos, o que diminui rapidamente a carga da bateria, ou mesmo, impede o alcance de metas de eficiência energética. Portanto, alguns dispositivos são compatíveis apenas com redes de rádio de baixa potência e com largura de banda limitada.

Em (TSCHOFENIG et al., 2015), são levantados os padrões de comunicação utilizados pelos desenvolvedores em ambientes de objetos inteligentes. Assim pode-se citar o padrão de comunicação *Device-to-Device*, o padrão *Device-to-cloud*, o padrão *Device-to-Gateway* (utilizado na abordagem da solução desenvolvida por este trabalho) e, por fim, o padrão *Back-End Data Sharing*.

O padrão *Device-to-Device* diz respeito a comunicação entre dispositivos, que podem ser heterogêneos (TSCHOFENIG et al., 2015). Esse modelo de comunicação exige que mesmo dispositivos de diferentes fabricantes consigam concordar com a pilha de protocolos implementada. Dentre os aspectos principais de *design* estão a tecnologia da camada física suportada (por exemplo *Bluetooth*), a versão de endereçamento (*Internet Protocol Version 4* (IPv4) ou *Internet Protocol Version 6* (IPv6)), como os dispositivos reconhecem um ao outro no meio de transmissão, qual protocolo de transporte utilizado (*User Datagram Protocol* (UDP) ou *Transmission Control Protocol* (TCP)), qual o protocolo da camada de aplicação utilizada (por exemplo *Constrained Application Protocol* (CoAP)), qual o método de representação de dados utilizados na comunicação e como os dados são codificados em termos de segurança no transporte.

Device-to-cloud se refere ao modelo de comunicação onde os dispositivos se conectam diretamente com o serviço de nuvem pela Internet para trocar dados e ser gerenciado (TSCHOFENIG et al., 2015). Algumas vezes o fornecedor das aplicações vende os objetos inteligentes, como pode ser o caso de alguma aplicação muito específica, e assim a comunicação pode ser direta sem necessidade de interoperabilidade (TSCHOFENIG et al., 2015). Desta forma, os dispositivos geralmente utilizam para a conexão redes IP do tipo *Wireless Local Area Network* (WLAN), Ethernet, Wi-Fi e redes celulares.

Device-to-Gateway é mais indicada quando tecnologias de rádio menos potentes são necessárias, como o caso do padrão IEEE 802.15.4, ou quando uma funcionalidade nova deve ser fornecida pela camada de aplicação, ou ainda quando é necessária a interoperabilidade com dispositivos heterogêneos e legados sem suporte a tecnologia IP (TSCHOFENIG et al., 2015). Nestes casos um *gateway* é posicionado para lidar com as funcionalidades necessárias e intermediar os processos de comunicação.

Back-End Data Sharing é um modelo de comunicação onde as redes de dispositivos não se conectam a apenas um provedor de serviços de aplicação, mas com vários (TSCHOFENIG et al., 2015). Esse padrão define a possibilidade de dispositivos servirem diversas aplicações e serviços de nuvem, de modo que possam analisar e combinar dados de várias fontes.

Nestes modelos, os dispositivos IoT são capazes de interagir com a rede utilizando alguns dos diversos protocolos da camada de rede existentes. Os protocolos NFC e *Bluetooth* são tecnologias de comunicação sem fio baseada em radio frequência que são eficientes para trocas de dados entre dispositivos em curto alcance (FENG et al., 2014; DECUIR, 2014).

O padrão 802.15.4 IEEE, foi desenvolvido para dispositivos e sensores que tenham restrições de consumo de energia, que se comuniquem por meio de rajadas curtas de tráfego de mensagem em distâncias curtas em uma rede *Low-Rate Wireless Personal Area Network* (LR-WPAN) (ANDERSON, 2017). Enquanto que, o padrão Wi-Fi 802.11ah, também chamado Wi-Fi *HaLow*, é uma tecnologia desenvolvido pela *WiFi Alliance* usada para conexão em redes sem fio de longa distância (RUTRONIK, 2021). Desta forma, além de protocolos específicos IoT, as tecnologias de redes celulares, como a 2G/3G/4G/5G, as redes convencionais Wi-Fi e cabeadas também podem ser utilizadas para a comunicação dos dispositivos IoT (KROODO, 2019).

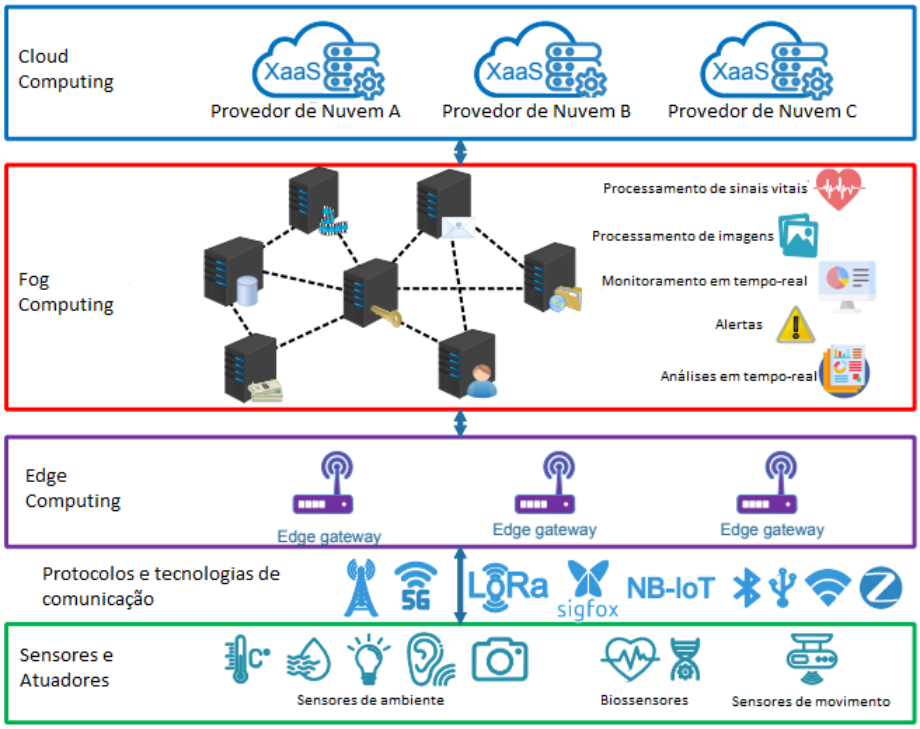
Os dispositivos ainda podem utilizar variados tipos de protocolos da camada de aplicação. Dispositivos convencionais, como os *smartphones*, podem utilizar protocolos comuns da rede IP como HTTP e *File Transfer Protocol* (FTP). No entanto, dispositivos IoT geralmente utilizam versões modificadas ou específicas desses protocolos, como *Message Queue Telemetry Transport* (MQTT), *Representational State Transfer* (REST) API e CoAP (utiliza UDP) que são exemplos de protocolos utilizados por dispositivos que possuem suporte ao conjunto de protocolos TCP/IP (NGUYEN-HOANG; VO-TAN, 2019).

Esse grande número de protocolos remete a situações de conversão de protocolos e isso exige o uso de dispositivos intermediários como *gateways*. Um dispositivo *gateway* pode ser qualquer dispositivo com recursos computacionais que permitam realizar os processos de comunicação e de processamento exigidos. De acordo com (TADINADA, 2014; COSTA; OLIVEIRA, 2020), dentre algumas das funcionalidades mais importantes que podem ser implementadas por *gateways* estão :

- Servir como um *sink node* agregando dados e os encaminhando para outros dispositivos ou para a Internet.
- Proteger os dispositivos de ataques externos.
- Identificar tipo de tráfego e identificar requisitos de QoS.
- Identificar prioridade de tráfego.

Por outro lado, os servidores de aplicações para cidades inteligentes podem estar um pouco distantes da borda da rede onde os dispositivos geradores de dados estão (Figura 4). As aplicações são desenvolvidas sobre ambientes de nuvem computacional, que são caracterizadas por oferecer uma infraestrutura muito grande e elástica de recursos virtualizados, para armazenamento, processamento e comunicação (SANTANA et al., 2017). Esses conceitos são fundamentais para implementação de sistemas para cidades inteligentes.

Figura 4 – Infraestrutura típica de Cidades Inteligentes.



Fonte: (PALARMINI, 2020).

Assim, juntamente com a nuvem computacional, outras soluções como computação em névoa (Fog Computing) também podem ser utilizadas para processar os dados para as aplicações de cidades inteligentes (SANTANA et al., 2017). Esse modelo é importante pois traz algumas características da nuvem computacional para mais próximo da borda da rede, diminuindo o atraso de transmissão de dados entre plataformas e dispositivos.

Neste sentido, para realizar as interconexões entre dispositivos geradores de dados, ou *gateways*, e os sistemas de nuvem, pode-se implementar redes baseadas na arquitetura SDN. As redes atuais (legado) não suportam o gerenciamento dinâmico necessário de QoS e priorização que os fluxos das cidades inteligentes exigem (Zhang et al., 2018).

2.1.4 Aplicações e Tipos de Serviços

As plataformas de cidades inteligentes devem suprir quatro domínios de aplicações principais, sendo eles, aplicações relacionadas a negócios, relacionadas aos cidadãos, relacionadas aos ambientes e relacionadas ao governo (YIN et al., 2015). O Anexo A apresenta esses 4 domínios de aplicação e seus subdomínios de atuação. É esperado que uma *smart city* utilize os recursos computacionais e recursos de comunicação, integração, gerenciamento e análise da grande quantidade de dados gerada pelos componentes para melhorar a segurança, eficiência, produtividade e qualidade de vida para os cidadãos (ALAVI et al., 2018).

O domínio de aplicações relacionadas a negócios envolve melhorar a eficiência e qualidade da gestão interna de uma organização, visando a torná-la mais próspera (YIN et al., 2015).

Esta categoria compreende os seguintes subdomínios: anúncios e propagandas, agricultura, empreendedorismo, gerenciamento de empresas, logísticas e transações.

No domínio de aplicações para os cidadãos, estão relacionados principalmente as subáreas que visam melhorar o aproveitamento da vida e a felicidade das pessoas (YIN et al., 2015). Entre os subdomínios estão as aplicações destinadas à educação, entretenimento, saúde, transporte público, tráfego inteligente e turismo.

O terceiro domínio definido em (YIN et al., 2015), é voltado para as aplicações que interagem e dependem do ambiente ao qual estão atuando. São aplicações que buscam a sustentabilidade por meio dos dados sobre o ambiente e do comportamento dos cidadãos. Por isso, este domínio abraça o desenvolvimento nas áreas de construção, habitação, espaços públicos, energias renováveis, *smart grid*, controle da poluição, gestão de resíduos e gestão da água.

Por último, o domínio de aplicações destinadas ao governo e administração, busca compreender as aplicações destinadas a tornar a cidade mais eficiente, no sentido de ser mais transparente, com melhor segurança e eficácia em serviços prestados aos cidadãos (YIN et al., 2015). Este domínio compreende as seguintes categorias: monitoramento das cidades, *e-government*, resposta a emergências, segurança pública, serviços públicos e governo transparente.

Dentro dessas categorias, aplicações de vídeo monitoramento são algumas das que possuem maior necessidade de recursos de rede e variação de prioridade. Esse tipo de aplicação exige detecção e rastreamento de objetos em tempo real por processamento de fluxos de vídeo, que devem ser coletados de diversas fontes, como câmeras e dispositivos móveis inteligentes, transferindo uma enorme quantidade de quadros de vídeo em direção a um serviço de nuvem (NIKOUEI; CHEN; FAUGHNAN, 2018).

Vídeo monitoramento baseado em câmeras policiais, corporais ou veiculares, é um exemplo de aplicação inteligente (HUANG; CHOU; WU, 2021). Nesse caso, veículos policiais possuem um dispositivo que recebe os dados das câmeras policiais, funcionando como um *sink node* e são capazes de se conectar a rede sem fio da cidade, por algum ponto Wi-Fi ou rede móvel, para enviar video em tempo-real para uma central de monitoramento.

Atualmente, as operações de segurança e emergência na maioria das cidades dependem de redes de comunicação desconectadas (KASHEF; VISVIZI; TROISI, 2021). Durante emergências, as redes convencionais ficam sobrecarregadas e tornam-se incapazes de fornecer serviços adequados. Deste modo, se faz necessários novos mecanismos de reserva de recursos e priorização para fluxos de cidades inteligentes que possam auxiliar nesses cenários.

2.2 QUALIDADE DE SERVIÇO

Com a crescente demanda das cidades inteligentes, os fluxos de dados envolvendo seus dispositivos começam a precisar cada vez mais de recursos de rede. Segundo (VAILSHERY, 2021), em 2025 a quantidade de dispositivos IoT conectados deve atingir a marca de 30.9 bilhões, enquanto que os dispositivos não IoT conectados devem chegar a 10 bilhões. Esses

dispositivos geram e consomem fluxos de dados que possuem diferentes requisitos de qualidade, o que impõe muita demanda na rede e exigem novos mecanismos de gerenciamento de recursos (KHANVILKAR et al., 2004). Um exemplo desse tipo de fluxos são de aplicações de tempo real, como videoconferência, as quais são muito sensíveis ao atraso de transmissão e *jitter*, necessitando de certa largura de banda garantida (ITU, 2001a).

A qualidade de serviço se define pelas características de um serviço de telecomunicação que garantem a capacidade de satisfazer as necessidades do usuário do serviço, tanto dos requisitos anunciados quanto dos implícitos (JANEVSKI; JANKOVIC; MARKUS, 2017). Nas redes convencionais, os fluxos de aplicações disputam os recursos disponíveis na rede. Quando esses recursos não são administrados, não se pode garantir a sua distribuição apropriada, fazendo com que tráfego de maior prioridade (mais importante) seja prejudicado (KARAKUS; DURRESI, 2017). Desta forma, para que se possa fornecer os recursos necessários ao fluxo de dados de um serviço é necessário conhecer seus requisitos, bem como identificar a quantidade de recursos que deve ser alocada.

A infraestrutura de rede da Internet atual não se compromete com garantias de entrega nem de QoS, apenas implementa um sistema de entrega *best-effort*. A Internet é formada por grupos de *Autonomous System* (AS)s independentes (JANEVSKI; JANKOVIC; MARKUS, 2017). Cada sistema autônomo é responsável pelo gerenciamento de tráfego e uso de protocolos de roteamento dentro de seu domínio. Por este motivo, garantir QoS é uma tarefa complexa em ambientes baseados na Internet. No entanto, o interesse em aplicações cada vez mais modernas e o surgimento de novos paradigmas de rede, como SDN, buscam mudar esse panorama.

O tráfego de dados na Internet é basicamente de aplicações multimídia, ou seja, uma combinação de dados (texto), vídeo e áudio. Aplicações de dados estão distribuídas pela Internet de diversas formas, como páginas HTML e outros tipos de arquivos para *download*. Os requisitos de largura de banda para aplicações deste tipo dependem muito do tamanho dos arquivos que são requisitados. Além disso, podem suportar algum erro ou perda de pacotes dependendo do tipo de aplicação, por exemplo aplicações de *chat* de mensagens instantâneas, ou então, não tolerar perdas, como no caso de transferências de arquivos (KHANVILKAR et al., 2004). Por outro lado, aplicações do tipo áudio e vídeo dependem da qualidade e do método de compressão envolvido para determinar seus requisitos de QoS, além de geralmente possuírem tolerância a perda ou erros de pacotes.

Dentro dessas categorias, ainda se pode ter tráfego de dois tipos: mídias de tempo-real e de não-tempo-real (KHANVILKAR et al., 2004). Essa classificação se refere aos requisitos de tempo que a entrega de pacotes de fluxos de tempo-real necessitam com relação à atraso e *jitter*, de modo que fluxos não-tempo-real não possuem esses requisitos.

Sabendo disso, surgiram diversos estudos sobre os requisitos de qualidade mais comuns para os fluxos de dados da Internet (HARYADI, 2013; KHANVILKAR et al., 2004; ITU, 2001a; KRUGER, 2017; CHEN; FARLEY; YE, 2004). Esses estudos foram propostos como forma de recomendações e orientações sobre os fatores que influenciam na QoS e quanto de cada fator

se faz necessário. A Tabela 1 comprime alguns desses estudos levando em consideração tipo de tráfego, qual aplicação, necessidade de taxa de transferência de dados, o atraso e a perda tolerável de pacotes para esse tipo de tráfego. Da mesma forma, as Tabelas 2, 3, 4, comprimem estudos sobre os requisitos de tráfego de dispositivos geradores de dados de cidades inteligentes, levando em conta tipo de tráfego, aplicação, tamanho da rede, taxa de transferência e atraso tolerável.

Tabela 1 – Aplicações e Requisitos de QoS.

Tipo de tráfego	Aplicação	Taxa de transferência	Atraso tolerável (*One-way*)	Perda de pacotes
Áudio	VOIP	Baixa qualidade - 4 kbit/s Média qualidade - 32 kbit/s Alta qualidade - 64 kbit/s	<150ms	<3%
Áudio	Mensagem de voz	Baixa qualidade - 4 kbit/s Média qualidade - 16 kbit/s Alta qualidade - 32 kbit/s	<400ms	<3%
Áudio	*Streaming* de áudio com alta qualidade	Baixa qualidade - 16 kbit/s Média qualidade - 64 kbit/s Alta qualidade - 128 kbit/s	<150ms	<1%
Vídeo	Chamada de vídeo	SD-1 Mb/s HD-2 Mb/s	<150ms	<1%
Vídeo	Transmissão de vídeo local/tempo-real	SD-4 Mb/s HD-8 Mb/s 4K-25 Mb/s	<150ms	<1%
Dados	*Web-browsing* - HTML	1 Mb/s	<400ms	0%
Dados	Jogos multijogador	2 Mb/s 4 Mb/s	<150ms	< 3%
Dados	*Download* de arquivos	2 Mb/s 10 Mb/s 20 Mb/s	<400ms	0%
Dados	*Upload* de arquivos	2 Mb/s 10 Mb/s 20 Mb/s	<400ms	0%
Dados	Comando e controle	1 Mb/s	<400ms	0%
Dados	Imagem	1 Mb/s	<400ms	0%
Dados	*E-mail*	1 Mb/s	<400ms	0%
Dados	Mensagens de texto	1 Mb/s	<400ms	0%
Dados	Controle de rede	1 Mb/s	<150ms	0%

Fonte: Adaptado de (ITU, 2001a; KRUGER, 2017; AUTHORITY, 2018; CHEN; FARLEY; YE, 2004; MAINUDDIN; DUAN; DONG, 2021; MOCNEJ et al., 2018).

2.2.1 Priorização de Serviços em Cidades Inteligentes

Os cenários de cidades inteligentes favorecem o desenvolvimento de aplicações por meio do uso, principalmente, de sensores e atuadores. No entanto, as aplicações podem ter diferentes níveis de criticidade de acordo com a relevância dos dados medidos e das aplicações para as quais esses dados são utilizados e devem ser tratadas de diferentes formas na rede (COSTA et al., 2015).

Em algumas dessas aplicações, estão incluídos serviços que possuem mais impacto nas tarefas que executam do que outros (COSTA; OLIVEIRA, 2020). Por exemplo, as aplicações que envolvem a proteção e integridade das pessoas. Por isso, os objetivos dos fluxos de dados nestes

Tabela 2 – Características de Tráfego de dispositivos IoT - Casas Inteligentes.

Tipo de tráfego	Aplicação	Tamanho da rede	Taxa de transferência	Atraso tolerável
Dados	Monitoramento das condições da construção	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego irregular, uma mensagem a cada hora por dispositivo	<400ms
Dados	Controle de iluminação	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego irregular e infrequente	<400ms
Dados	Detecção de anomalias (incêndio, gás, sujeira...)	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego irregular e infrequente	<400ms
Dados	Controle de temperatura interna	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego regular, uma mensagem a cada 15 min por dispositivo	<400ms
Dados	Aparelhos inteligentes	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego irregular e infrequente.	<400ms
Dados	Monitoramento do uso de água e energia	Em torno de dez dispositivos	500 kb/s Tráfego irregular e infrequente.	<400ms
Vídeo	Monitoramento de vídeo (interno e externo)	Em torno de uma dezena de dispositivos	As mesmas taxas de transmissão de vídeo em tempo-real. Tráfego em intervalos regulares.	<150ms

Fonte: Adaptado de (MOCNEJ et al., 2018).

Tabela 3 – Características de Tráfego de dispositivos IoT - Hospitais Inteligentes.

Tipo de tráfego	Aplicação	Tamanho da rede	Taxa de transferência	Atraso tolerável
Dados	Gerenciamento de doenças crônicas	<10 dispositivos	500 kb/s Tráfego irregular, uma mensagem a cada hora por dispositivo	<150ms
Dados	Monitoramento de condição pessoal	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego irregular e infrequente	<400ms
Dados	Detecção de anomalias (incêndio, gás, sujeira...)	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego irregular e infrequente	<400ms
Dados	Controle de temperatura interna	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego regular, uma mensagem a cada 15 min por dispositivo	<400ms
Vídeo	Monitoramento dos pacientes	Entre dezenas e centenas de dispositivos	As mesmas taxas de transmissão de vídeo em tempo-real. Tráfego em intervalos regulares.	<150ms
Dados	Monitoramento do uso de água e energia	Em torno de dez dispositivos	500kb/s Tráfego irregular e infrequente.	<400ms
Vídeo	Monitoramento de vídeo (interno e externo)	Dezenas de dispositivos	As mesmas taxas de transmissão de vídeo em tempo-real. Tráfego em intervalos regulares.	<150ms

Fonte: Adaptado de (MOCNEJ et al., 2018).

cenários podem ser analisados, de forma que possam ser tratados com algum tipo de priorização dependendo do nível de criticidade dos sensores e serviços envolvidos.

Muitos parâmetros de priorização podem ser usados para diferenciar os dados (COSTA et al., 2015). Eles podem ser diferenciados pelo seu tipo (áudio, vídeo ou outro), pelos requisitos de tempo, pela criticidade e pela confiabilidade.

O conceito de priorização pode ser interpretado como uma característica de qualidade de serviço (COSTA; OLIVEIRA, 2020). Assim, um dos desafios dessa abordagem é identificar adequadamente as diferentes prioridades e adaptar a rede para corresponder a essa exigência.

A dinâmica natural das cidades resulta em um comportamento constante e previsível, são eventos adversos, como um acidente de carro, um engarrafamento ou um incêndio, que definem a mudança de estado e a prioridade associada aos serviços envolvidos. Desta forma, (COSTA; OLIVEIRA, 2020) classificou as aplicações das cidades inteligentes em três grupos, conforme a sensibilidade aos eventos críticos que devem mudar a prioridade das aplicações:

Tabela 4 – Características de Tráfego de dispositivos IoT - Prefeituras Inteligentes.

Tipo de tráfego	Aplicação	Tamanho da rede	Taxa de transferência	Atraso tolerável
Dados	Monitoramento das condições da construção	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego irregular, uma mensagem a cada hora por dispositivo	<400ms
Dados	Controle de iluminação	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego irregular e infrequente	<400ms
Dados	Detecção de anomalias (incêndio, gás, sujeira...)	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego irregular e infrequente	<400ms
Dados	Controle de temperatura interna	Entre dezenas e centenas de dispositivos	500 kb/s Tráfego regular, uma mensagem a cada 15 min por dispositivo	<400ms
Dados	Monitoramento do uso de água e energia	Em torno de dez dispositivos	500kb/s Tráfego irregular e infrequente.	<400ms
Vídeo	Monitoramento de vídeo (interno e externo)	Dezenas de dispositivos	As mesmas taxas de transmissão de vídeo em tempo-real. Tráfego em intervalos regulares.	<150ms

Fonte: Adaptado de (MOCNEJ et al., 2018).

- Aplicações que não reconhecem eventos: seguem um padrão de funcionamento que não varia com alterações ambientais.
- Aplicações de monitoramento de eventos: aplicações projetadas para monitorar a ocorrência de eventos específicos, por exemplo sensores de incêndio.
- Aplicações reativas a eventos: não são projetadas para monitorar eventos mas são afetados pela sua ocorrência. Por exemplo, câmeras de monitoramento que flagram um acidente entre dois veículos.

Em (COSTA; OLIVEIRA, 2020) foi estabelecido um modelo de gerenciamento para lidar com os diferentes tipos de dados fornecidos pelos sensores, para otimizar o desempenho das aplicações de cidade inteligente bem como as concorrências entre esses tipos de aplicações por meio da priorização. Neste cenário, os dados coletados por dispositivos IoT eram diferenciados utilizando um componente conectado a eles, chamado de módulo central de processamento. Este módulo serve como *gateway* e possui a função de armazenar tabelas de priorização, que estão de acordo com os valores definidos pelas aplicações que fazem uso desses dados. Assim, os dados dos dispositivos IoT que passam por esse módulo são testados nas tabelas de priorização para se identificar o nível de criticidade dos valores medidos.

No entanto, existem outras formas de identificação de prioridade que podem ser implementadas. A priorização está centrada na relevância dos dados coletados para a aplicação e na relevância dessa aplicação (COSTA et al., 2015). Desta forma, as aplicações podem identificar essa criticidade e utilizar meios de informar a rede para obter o QoS desejado.

2.2.2 Acordos de QoS

Os emissores de dados, conhecendo seus requisitos de QoS, precisam utilizar mecanismos de negociação com a rede, para que estes requisitos sejam garantidos. Neste sentido, a Internet convencional possui alguns protocolos para negociação de QoS, como por exemplo *Multiprotocol*

Label Switching (MPLS) (traduzido como comutação multiprotocolo por rótulo) e *Service Level Agreenment* (SLA) (traduzido como acordo em nível de serviço).

O MPLS é um protocolo de roteamento baseado em pacotes rotulados. O cabeçalho MPLS é posicionado entre o cabeçalho da camada 2 e antes do cabeçalho da camada 3, e contém o rótulo que é usado para identificar os pacotes em cada roteador. Os rótulos são usados para classificar os pacotes em classes de encaminhamento equivalentes. Assim, com base nos rótulos dos pacotes os roteadores identificam as rotas e o próximo salto, remarcando o pacote para que este tenha sentido no próximo roteador (ROSEN A. VISWANATHAN, 2001). Além do campo que contém o valor do rótulo, o cabeçalho MPLS contém outras informações (CRISTIANE, 2014):

- Campo rótulo: Contém 20 bits e leva o valor do rótulo atribuído.
- Campo EXP: Campo de 3 bits, que define a classe de serviço de um pacote, disponibilizado para questões de priorização.
- Campo S (*stack*, traduzido como pilha): Campo de 1 bit, que serve para indicar quando é utilizado mais de um rótulo no pacote.
- Campo TTL (*Time To Live*, traduzido como tempo de vida): Este campo tem o mesmo papel que o campo TTL do cabeçalho IP. Possui 8 bits e serve para indicar quantos saltos o pacote realizou.

Nos roteadores, esses rótulos podem ser programados e gerenciados pela operadora de rede responsável. No entanto, MPLS é destinado para escolha de rotas e não reserva de recursos. Por outro lado, o mecanismo SLA é um modelo de acordo formal entre duas ou mais entidades para garantir características de entrega de dados entre as partes. O SLA não é exatamente um mecanismo para implementar QoS, mas é uma espécie de contrato que define os entendimentos mútuos com relação aos recursos de rede desejados entre as duas partes (JIN; MACHIRAJU; SAHAI, 2002). Assim, um SLA pode conter diversos componentes, entre eles:

- Propósito: Descreve o motivo da criação do SLA.
- Partes: Descreve as partes envolvidas no SLA, para definir quem é cliente e quem é provedor.
- Período válido: Define por quanto tempo o contrato é válido.
- Restrições: Define quais são as ações envolvidas para que os serviços solicitados sejam prestados.
- Objetivos em nível de serviço: Definem o conjunto de indicadores que se espera, como desempenho, confiabilidade e outros aspectos.

- Penalidades: Definem o que acontece caso o provedor de serviços não cumpra com o estabelecido.

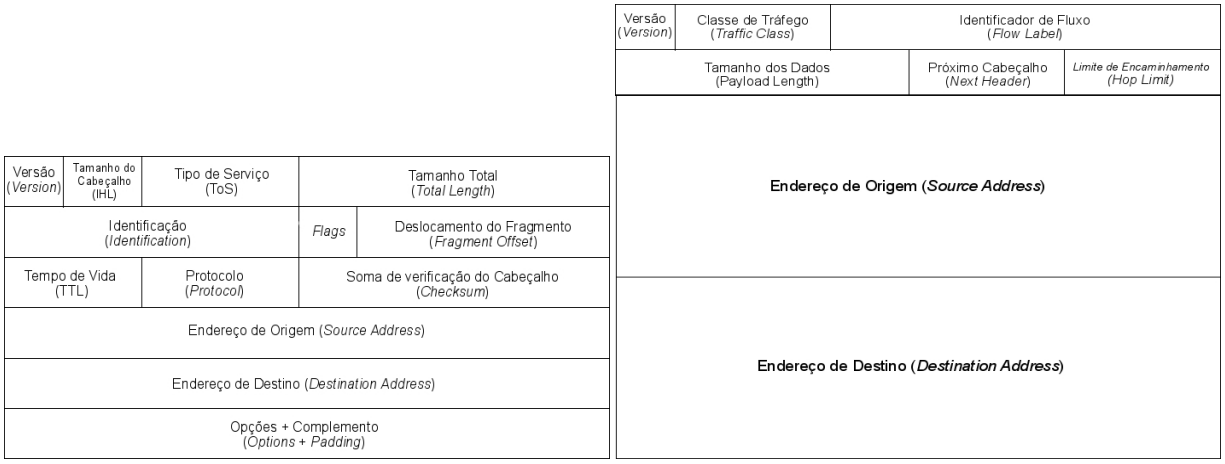
A grande questão desses modelos é o quanto se pode prometer e o que se pode entregar em termos de QoS em redes de Internet convencionais. Assim, o uso de redes SDN é essencial para a implementação de QoS, onde é possível programar o comportamento de cada dispositivo de rede de forma dinâmica e se pode implementar novos mecanismos para descobrir os requisitos de QoS envolvidos no transporte dos fluxos de dados.

2.2.3 Diferenciação de Serviços

Os acordos entre emissores e provedores de rede permitem descobrir os requisitos de QoS para os fluxos de dados que serão gerados pelo emissor. No entanto, o provedor de rede necessita de meios para identificar esses fluxos na rede. O número de dispositivos IoT conectados a Internet é muito grande e atualmente são utilizado duas formas de endereçamento, IPv4 e IPv6, que possuem diversos campos de cabeçalho que podem auxiliar nesse processo.

Uma das formas de diferenciar os pacotes de fluxos de dados e identificar seus requisitos é classificar os tipos de tráfego em classes e marcar os pacotes dos fluxos com valores correspondentes a essas classes nos dispositivos de rede. Para isso, no *Request for Comments* (RFC) 791 (WAY, 1981), foi definido um campo no cabeçalho IPv4 com 8 bits, denominado *Type of Service* (TOS) ou tipo de serviço em português, posteriormente nomeado como *Differentiated Services* (DiffServ) no RFC 2474 (NICHOLS et al., 1998), conforme (CISCO, 2008). No IPv6 (DEERING; HINDEN, 1998), o campo também possui 8 bits e serve para o mesmo propósito, mas se chama *Traffic Class*. No entanto, apenas os 6 primeiros bits do campo devem ser utilizados, se chamando campo *Differentiated Services Code Point* (DSCP). Os dois últimos bits são reservados para usos futuros. A Figura 5 apresenta os cabeçalhos IPv4 e IPv6 com seus respectivos campos, incluindo os campos TOS e *Traffic Class*.

Figura 5 – Cabeçalho IPv4 e cabeçalho IPv6.



Fonte: (IPV6.BR, 2022).

Os bits válidos podem ser preenchidos por 0 (bits zero), significando que o campo não é utilizado pelo protocolo, ou podem ser utilizados para marcar, classificar ou medir características do pacote, que serão interpretadas pelos roteadores compatíveis da rede. Esse campo não tem utilidade na maioria dos casos na rede convencional devido a limitação no tratamento por fluxo de forma dinâmica, entretanto, pode ser explorado em redes baseadas em SDN por meio de estratégias configuradas pelo controlador.

O campo de diferenciação de serviço tem a utilidade de marcar os pacotes conforme a classificação do serviço que corresponde (Tabela 5). Uma classe de serviço representa uma categoria de fluxos de dados que necessita de características específicas da rede, como banda e atraso (BABIARZ K. CHAN, 2006). Essas classes definem o comportamento em cada salto (*switch* ou roteador) para os fluxos marcados. Desta forma, esse mecanismo exige que os dispositivos de rede tenham estratégias de marcação e identificação desses pacotes.

No (BABIARZ K. CHAN, 2006), foi proposto utilizar os 6 *bits* disponíveis para definir três comportamentos de encaminhamento em redes convencionais. Cada método de encaminhamento foi dividido em três níveis de prioridade (ouro, prata e bronze), conforme Tabela 5:

- Encaminhamento padrão (*Default Forwarding* - DF): Este comportamento se refere ao serviço *best-effort* (melhor esforço), aceitando os pacotes com alguma garantia de largura de banda. Pacotes marcados com DF podem ser perdidos, reordenados, duplicados ou atrasados.

- Encaminhamento garantido (*Assured Forwarding* - AF): Esse comportamento é destinado para redes que oferecem acordos de nível de serviços (SLA). É um serviço de melhor esforço aprimorado. Nele, o receptor detecta perda ou jitter e fornece *feedback* para que o remetente ajuste a taxa de transmissão de acordo com a capacidade disponível na rede.

- Encaminhamento acelerado (*Expedited Forwarding* - EF): A intenção dessa categoria é fornecer recursos para serviços que necessitem baixa perda de pacotes, baixo atraso e baixo jitter. Usando técnicas de enfileiramento, a probabilidade de atraso ou variação é minimizada, desde que se policie essas garantias.

- Seletor de classe (*Class Selector* - CS): Não define exatamente um comportamento, mas fornece compatibilidade com modelos de marcação que vieram antes do DiffServ, que marcavam o campo com outros valores, como é o caso do *IP Precedence* (Precedência IP).

Cabe ressaltar, que os valores desse campo do cabeçalho podem ser alterados ao longo do caminho entre origem e destino. Algumas aplicações hoje em dia definem o próprio DSCP. Além disso, os ASs distribuídos pela Internet podem colocar valores que façam sentido apenas internamente ou ainda apagar os valores desse campo (BARIK et al., 2019).

Tabela 5 – Valores DSCP.

Valor DSCP	Significado	Descrição	Probabilidade de descarte
000 000	BF	Melhor esforço	N/A
001 010	AF11	Classe 1, Ouro	Baixa
001 100	AF12	Classe 1, Prata	Media
001 110	AF13	Classe 1, Bronze	Alta
010 010	AF21	Classe 2, Ouro	Media
010 100	AF22	Classe 2, Prata	Baixa
010 110	AF23	Classe 2, Bronze	Alta
011 010	AF31	Classe 3, Ouro	Baixa
011 100	AF32	Classe 3, Prata	Media
011 110	AF33	Classe 3, Bronze	Alta
100 010	AF41	Classe 4, Ouro	Baixa
100 100	AF42	Classe 4, Prata	Media
100 110	AF43	Classe 4, Bronze	Alta
001 000	CS1	Classe 1	
010 000	CS2	Classe 2	
011 000	CS3	Classe 3	
100 000	CS4	Classe 4	
101 000	CS5	Classe 5	
110 000	CS6	Classe 6	
111 000	CS7	Classe 7	
101 110	EF	Encaminhamento rápido	N/A

Fonte: Adaptado de (CISCO, 2016; SOPHOS, c2018).

Para conservar os valores do cabeçalho existem mecanismos de tunelamento e mecanismos de acordo entre ASs. Tunelamento consiste em encapsular um pacote dentro de outro, de forma que o pacote encapsulado fique visível apenas para o emissor e o receptor (VPNMENTOR, 2022). Já mecanismos de acordo, envolvem negociar com os ASs do caminho entre origem e destino, para que a marcação de determinado pacote não seja alterada e que as mesmas políticas de QoS sejam aplicadas.

2.2.4 Gerenciamento de Largura de Banda

O principal recurso que pode ser fornecido por dispositivos de rede é a largura de banda. Uma das técnicas mais utilizadas para gerenciamento de largura de banda consiste em limitar o tráfego de pacotes de um determinado fluxo em um dispositivo de rede. A prática é conhecida como *Traffic Shaping* (TS) (modelagem de tráfego). Esta técnica permite controlar a quantidade de recursos máxima que um fluxo consome, permitindo melhorar a QoS por meio do fornecimento de diferentes quantidades de recursos para cada fluxo (AL-HADDAD et al., 2021).

O método de limitação do tráfego que sai de um *switch* pode seguir um modelo de alocação de largura de banda (*Bandwidth Allocation Method* (BAM)). Um modelo BAM se

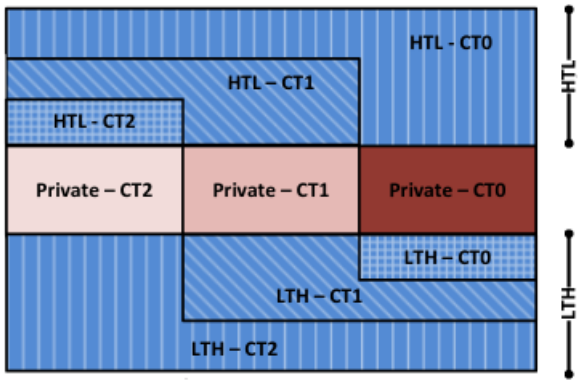
refere a estratégias de alocação de recursos, focadas principalmente em largura de banda, que permite o planejamento e agrupamento dos fluxos de dados com requisitos comuns em classes de serviços, fornecendo também, estratégias de compartilhamento de largura de banda entre classes (TORRES et al., 2020). As classes podem possuir um nível de prioridade, que serve para diferenciar fluxos mais importantes e fluxos menos importantes.

As principais estratégias de alocação de largura de banda seguem três modelos (TORRES et al., 2020):

- Recurso privado (*Private*): Os recursos são alocados com uso exclusivo por uma classe de tráfego (CT).
- Compartilhamento "alta para baixa"(*High-To-Low* (HTL)): A banda alocada para CTs de maior prioridade que não esteja sendo usada, é compartilhada de forma temporária com CTs de menor prioridade.
- Compartilhamento "baixa para alta"(*Low-To-High* (LTH)): A banda alocada para CTs de menor prioridade que não esteja sendo utilizada, é compartilhada de forma temporária com CTs de maior prioridade.

O modelo *Generalized Bandwidth Allocation Model* (G-BAM) é um modelo que mixa todas as três estratégias de alocação de largura de banda (Figura 6). Permite uma partição ser privada e sem compartilhamento desses recursos, enquanto as outras partições compartilham largura de banda de modo HTL e LTH (MARTINS; REALE; BEZERRA, 2014b). Este modelo se faz útil especialmente quando se quer aproveitar do oportunismo que os modelos de compartilhamento de largura de banda HTL e LTH trazem, enquanto se quer reservar uma partição com recursos privados, por exemplo, usados para controle da rede e admissão de novos fluxos.

Figura 6 – Modelagem de recursos G-BAM.



Fonte: Adaptado de (MARTINS; REALE; BEZERRA, 2014b)

As estratégias de reserva de recursos podem seguir dois modelos para a reconfiguração da alocação de recursos, as abordagens *hard* e *soft* (MARTINS; REALE; BEZERRA, 2014a). A

primeira abordagem, assume que quando uma reconfiguração na alocação de largura de banda ocorrer, a rede deve ser modificada imediatamente, independente das consequências que irá causar para os aplicativos e tráfego relacionados. Por exemplo, se a largura de banda de uma classe de serviços não está sendo utilizada ela pode ser emprestada para outra classe alocar mais fluxos. Então, no caso de a classe que emprestou solicitar a largura de banda, pelo modo *hard*, o fluxo que está emprestando largura de banda seria imediatamente removido para a alocação do novo.

Já a outra abordagem, *soft*, busca reduzir os impactos negativos, liberando a largura de banda gradativamente, o que leva mais tempo (MARTINS; REALE; BEZERRA, 2014a). Desta forma, seguindo o exemplo anterior, o fluxo não seria removido imediatamente, mas gradativamente a largura de banda de um fluxo seria transferida para o outro.

Em redes convencionais este modelo de gerenciamento de largura de banda não pode ser implementado corretamente de modo dinâmico. No entanto, o paradigma de redes SDN permite aproveitar da visão global que o controlador possui, para obter informações dos estados dos *swithes*, bem como utilizar o controlador para administrar as funções necessárias de diferenciação de fluxos e compartilhamento de recursos.

2.2.5 Gerenciamento de Filas

Neste sentido, o método BAM pode ser implementado de modo que as classes de tráfego sejam gerenciadas por diferentes filas nas portas de saída dos dispositivos de rede. Além disso, com os mecanismos de agendamento fornecidos pelas filas, é possível implementar as funcionalidades de priorização para classes de tráfego, que é um requisito de fluxos de dados de cidades inteligentes.

As filas são mecanismos que armazenam pacotes de fluxos de dados até que estes sejam encaminhados. No Linux, o *Traffic Control* (TC) (Controle de Tráfego) é a ferramenta utilizada para configurar o controle de tráfego das interfaces de rede (KERRISK, 2010). O controle de tráfego consiste no agrupamento de quatro ações:

- Modelagem (*Shaping*): A modelagem ocorre na porta de saída do dispositivo de rede e diz respeito a limitar a largura de banda disponível.
- Agendamento (*Scheduling*): Agendar a transmissão permite melhorar a interatividade do tráfego e ainda garantir largura de banda para transferências de pacotes em massa (enfileiramento). Este mecanismo também é chamado de priorização e ocorre apenas na porta de saída.
- Policiamento (*Policing*): O policiamento diz respeito a tratar o tráfego que chega pela porta de entrada do dispositivo de rede.
- Descarte (*Dropping*): O tráfego que excede o limite de largura de banda pode ser descartado, tanto na entrada como na saída.

O componente *Queueing Disciplines* (qdisc) faz parte do TC. Cada qdisc é uma fila de saída de pacotes configurada em uma interface de rede. Quando o *kernel* precisa enviar um pacote para uma interface, este pacote é enfileirado em uma qdisc configurada nesta interface, e quando precisa retirar pacotes, ele retira da qdisc *root* (fila base) (KERRISK, 2010).

Toda interface tem suporte a dois tipos de qdisc, *root* que é a fila padrão de uma interface de rede e trata do tráfego egresso, e *ingress* que trata do tráfego ingresso. A qdisc de tráfego ingresso suporta apenas policiamento, que é mais limitado a restringir ou negar tráfego em uma interface de entrada. Já no tráfego egresso, se pode aplicar técnicas de agendamento de entrega e *shaping*, além de policiamento e descarte de pacotes.

No Linux se tem dois tipos de qdisc: *classless* e *classful*. Os qdisc do tipo *classless* não podem conter classes, diferente das qdisc do tipo *classful*, que podem. Essas classes são, na verdade, filas e definem uma forma de dividir o tráfego para um tratamento diferenciado, uma vez que permitem associar filtros e prioridades entre elas. Além disso, classes permitem adicionar novos qdiscs para montar filas com estruturas de árvore.

Alguns dos parâmetros que podem ser configurados pelas classes (KERRISK, 2010):

- Posicionamento da classe dentro da hierarquia, como por exemplo, um identificador para a classe e um para a sua classe pai.
- Prioridade em que a classe será testada. No processo *round-robin* de desenfileiramento, as classes com menor prioridade são testadas primeiro.
- Taxa máxima de entrega garantida que esta classe e seus filhos estão condicionados.
- Taxa máxima que uma classe pode enviar se a classe pai tiver largura de banda disponível (i.e., *ceil* - teto).
- Quantidade de *bytes* que podem chegar em rajada (*burst*), ou seja, tão rápido quanto a interface pode transmiti-los.
- Número de *bytes* que devem ser retirados da classe e que pertençam a um único fluxo, antes que o escalonador alterne para retirar de outra classe.

Um dos mecanismos para gerenciamento de filas *classful* é o qdisc balde de *tokens* hierárquicos (*Hierarchical Token Bucket* (HTB)) (REN, 2017). O HTB utiliza dois mecanismos para modelagem do desenfileiramento das filas: *tokens* (fichas) e *buckets* (baldes) (KERRISK, 2010). Os *buckets* definem o número de *tokens* que estão disponíveis para serem utilizados, e se referem a taxa de entrega associada a uma classe. Os *tokens* são gerados a uma taxa desejada e são usados pelos pacotes de uma fila para serem enviados e então são repostos.

No HTB, as classes são criadas contendo um qdisc (fila), por padrão do tipo *Fist In First Out* (FIFO). No entanto, para o desenfileiramento, elas são consultadas utilizando a sua

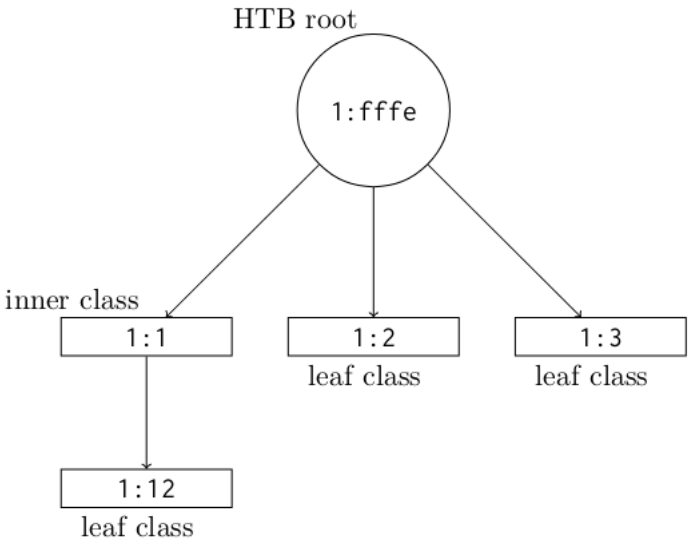
prioridade num processo *round-robin*. Ou seja, a priorização ocorre entre as classes e não internamente nas filas.

Assim, quando uma classe deve desenfileirar, os pacotes da fila consomem os *tokens* disponíveis e vão sendo enviados, deixando a fila (KERRISK, 2010). Se não houver *tokens*, o tráfego é enfileirado e aguarda a geração de novos *tokens*.

Neste modelo, classes que utilizam menos largura de banda do que lhe foi atribuída, possuem largura de banda ociosa. Assim, as demais classes podem utilizar essa largura de banda ociosa até o valor da taxa especificada pelo campo *ceil*, em um modelo de compartilhamento de recursos oportunista.

A Figura 7 representa uma estrutura em árvore de filas HTB. A classe 1:fffe é a classe *root*, que possui três classes filhas 1:1, 1:2 e 1:3. A classe 1:1 ainda possui outra filha, 1:12. Classes que não possuem filhas são chamadas classes folhas (*leaf class*), enquanto classes que possuem filhas são chamadas classes internas (*inner class*).

Figura 7 – Estrutura HTB.



Fonte: Adaptado de (KRISHNA; ADRICHEM; KUIPERS, 2016).

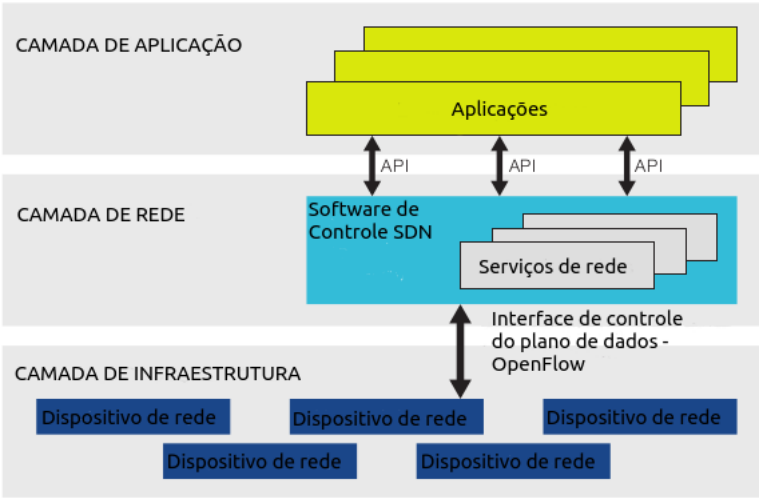
Esse tipo de modelagem permite diferenciar as classes em termos de prioridade. Com isso, cada classe HTB pode ser associada a uma classe de tráfego para aproveitar esse tipo de priorização e obter um modelo de prioridade de classes de tráfego baseado em filas. No sentido das cidades inteligentes, essa abordagem é muito promissora para tratar as prioridades dinâmicas dos fluxos de dados.

2.3 SOFTWARE DEFINED NETWORK

A arquitetura de redes definidas por *software* centraliza o gerenciamento da rede, permitindo programação do controle da rede tratando-a como uma entidade lógica e separando o plano de dados (encaminhamento dos *switches*) do plano de controle (lógica do controlador) (ONF,

2012). Desta forma, essa abordagem possibilita controlar, alterar e gerenciar o comportamento da rede dinamicamente através de interfaces de software (Krainyk; Dvornik; Krainyk, 2019).

Figura 8 – Visão simplificada da arquitetura SDN.



Fonte: Adaptado de (ONF, 2012)

A Figura 8 apresenta os principais componentes da arquitetura SDN. Assim, uma breve descrição de cada camada é dada em (KARAKUS; DURRESI, 2017):

- Camada de aplicação: Esta camada consiste em uma ou várias aplicações de rede, as quais utilizam da atuação do controlador para atingir os requisitos de QoS.
- Camada de controle: A camada de controle é composta por um ou mais controladores SDN programáveis, para fornecer acompanhamento e gerenciamento do fluxo de rede. Os controladores possuem interfaces que os conectam com as outras camadas, permitindo ampla atuação no gerenciamento.
- Camada de infraestrutura/dados: é o plano que consiste dos dispositivos de rede, como *switches* físicos/virtuais e pontos de acesso. Esses dispositivos são gerenciados pelos controladores SDN.

Implementar SDN facilita muito a instalação e modificação de dispositivos de rede por mover o controle para o *software* e centralizá-lo em um componente gerenciador (controlador SDN) (Krainyk; Dvornik; Krainyk, 2019). Isso permite explorar a capacidade de programação da rede dos controladores SDN para evitar reconfigurações físicas na infraestrutura da rede e reduzir o tempo de implementação de melhorias. Na rede tradicional, para adicionar um novo caminho entre um provedor de serviço e um usuário, novos roteadores seriam instalados. Já na arquitetura SDN, os controladores espalhados pela rede podem ser programados de modo a escolher um caminho específico para um determinado cliente ou classe de aplicação em trânsito. O trabalho de (Belgaum et al., 2019) cita algumas das funcionalidades que podem ser empregadas com SDN:

- Roteamento: É possível determinar rotas diferentes das tradicionais e de forma dinâmica.
- Balanceamento de carga: Com base em determinadas métricas, os pacotes podem ser encaminhados para outro *cluster* do mesmo serviço.
- Gerenciamento de recursos: Informações estatísticas podem ser coletadas e com base nelas, recursos como largura de banda podem ser gerenciados.
- *Green Computing*: O consumo de energia por meio dos dispositivos pode ser controlado, por exemplo controlando a necessidade de estarem ligados quando não há fluxo de dados.

Com o emprego de redes SDN em uma cidade inteligente, é possível desenvolver funcionalidades de gerenciamento utilizando o controlador, para fornecer QoS e priorização para seus fluxos de dados. Essa integração permite melhorar o desempenho das aplicações voltadas para cidades inteligentes, facilitando e incentivando a criação de novas cidades inteligentes e serviços.

2.3.1 Controlador

O controlador é a parte mais importante da arquitetura SDN, devido ao gerenciamento e controle que realiza sobre os equipamentos de comutação de pacotes - *switches* (Paliwal; Shrimankar; Tembhurne, 2018). Utilizando o protocolo de comunicação OpenFlow, o controlador é capaz de gerenciar diversos elementos dos *switches* por meio de troca de mensagens (TEAM, 2014).

Com esse protocolo, o controlador pode adicionar, atualizar e excluir entradas de fluxo nas tabelas internas dos *switches*, tanto de forma reativa quanto proativa (ONF, 2012). Assim, no modo reativo, quando um *switch* envia um pacote de dados para o controlador por não saber como lidar com ele devido a falta de regras relativas àquele tipo de pacote, o controlador pode examinar os cabeçalhos e o campo de dados desse pacote para tomar alguma decisão e configurar o *switch* com novas regras de encaminhamento. Já o modo proativo o processamento e instalação de novas regras ocorre de forma pré-programada, sem prévia interação a respeito de tipos de pacotes de dados com o(s) *switch(es)*.

Nos controladores também é possível implementar diversas ferramentas e aplicações de gerenciamento de rede. Como este elemento da arquitetura SDN possui visão global da rede, é muito efetivo em aplicar funcionalidade de redes que envolvam colaboração entre os elementos de rede.

Existem basicamente dois tipos de controladores, os centralizados e os distribuídos (Paliwal; Shrimankar; Tembhurne, 2018). Os centralizados implementam toda a lógica do plano de controle em um único local e dispositivo. Desta forma, os *switches* precisam ter conexão com esse único controlador. Essa abordagem é a mais simples de implementar devido a menor complexidade. Entretanto, o controlador se torna um ponto único de falha e pode sofrer com escalabilidade, devido à falta de capacidade de lidar com muitas conexões ou solicitações.

Os controladores distribuídos podem fornecer alta escalabilidade e desempenho, devido a possibilidade de tratar muitas requisições (Paliwal; Shrimankar; Tembhurne, 2018). Nesta abordagem, existem várias réplicas do controlador centralizado, mas distribuídos geograficamente. Cada *switch* está conectado ao melhor controlador na proximidade e em caso de falha, deve haver uma reconfiguração para se conectar em um controlador ativo.

Um controlador bastante conhecido é o Ryu. Conforme descrito em (Selvaraj; Nagarajan, 2017), ele é escrito totalmente em Python e está disponível sob licença Apache 2.0. Dentre os vários protocolos suportados para gerenciar os dispositivos de rede estão o OpenFlow, versão 1.0 até 1.5, Netconf (um protocolo para gerenciamento e configuração de rede) e OF-Conf (conjunto de configurações para comunicação controlador-*switch*).

2.3.2 Open Flow

O protocolo OpenFlow é chamado também de API *southbound* e foi a primeira interface de comunicação desenvolvida para a arquitetura SDN (ONF, 2012). Os *switches* e controladores se comunicam por meio de troca de mensagens do OpenFlow, usando interface de *sockets* convencional.

Cada versão do protocolo OpenFlow trouxe funcionalidades novas e mudanças quanto a versões anteriores (KARAKUS; DURRESI, 2017). A versão 1.0 permitiu que uma porta de saída pudesse implementar o gerenciamento de filas, com mais de uma fila para cada porta. Entretanto a versão 1.3 trouxe a funcionalidade mais distinta, que são as regras de medição (*Meter Table*). Com isso é possível medir fluxos e aplicar ações relacionadas com o valor das medidas, tornando possível comportamentos específicos para cada fluxo.

O protocolo OpenFlow pode ser implementado tanto em dispositivos reais de rede quanto nos controladores e *switches* SDN simulados por *software*. O OpenFlow usa o conceito de fluxos para identificar o tráfego de rede com base em regras de correspondência predefinidas, que podem ser programadas de modo estático ou dinâmico pelo controlador SDN (ONF, 2012). Desta forma, o fluxo do tráfego que passa pelos dispositivos pode ser controlado conforme a programação definida pelo controlador.

2.3.3 Open vSwitch

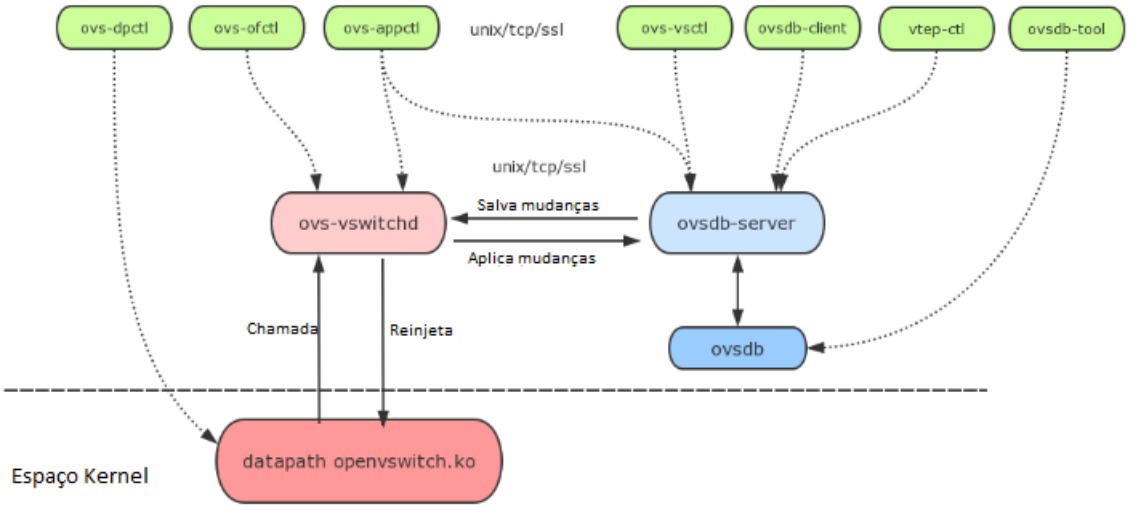
Um *switch* virtual que suporta o OpenFlow é o OVS, cuja arquitetura pode ser vista na Figura 9. Este componente consiste em um conjunto de tabelas de fluxo, tabela de grupo e canais de comunicação externos conectados a um controlador (ONF, 2012). Ele é escrito em C e é muito utilizado em ambientes de produção de larga escala. Além disso, essa ferramenta é adequada para trabalhar em ambientes de máquinas virtuais e oferece suporte a várias tecnologias de virtualização, como VirtualBox e *Kernel-based Virtual Machine* (KVM). Os componentes principais do OVS são:

- ovs-vswitchd: É um *daemon* interface que implementa o *switch* virtual. Este módulo

utiliza do *Kernel* do sistema operacional para acessar as interfaces de rede e realizar a comutação baseada em fluxo, ou seja, implementa o *switch* virtual.

- ovsdb-server: É um servidor de banco de dados que contém todas as configurações do *switch* virtual. A ferramenta ovs-vswitchd faz consultas nesse servidor para obter a configuração do *switch*.
- ovs-dpctl: É uma ferramenta para configurar o módulo de comutação do *switch*, e que executa em modo *kernel*.
- ovs-vsctl: Esta é uma ferramenta para consultar e atualizar as configurações do ovs-vswitchd, por meio de uma conexão com um ovsdb-server.
- ovs-appctl: Esta ferramenta fornece uma maneira simples de invocar comandos específicos por linha de comando para o ovs-vswitchd.
- ovs-ofctl: É um utilitário para consultar e controlar os *switches* OpenFlow e controladores.

Figura 9 – Arquitetura OVS.



Fonte: Adaptado de (SOBYTE, 2022)

O ovs-vswitchd trabalha junto com os *datapaths* (*switch* ou *bridge*) para implementar a comutação de pacotes baseada em fluxos (SOBYTE, 2022). Este componente pode se comunicar com controladores através do protocolo OpenFlow e se conectar ao banco de dados ovsdb-server utilizando o protocolo OVSDB.

Um ovs-vswitchd suporta vários *datapaths* independentes. Quando o ovs-vswitchd é inicializado, ele lê as informações do ovsdb-server para se configurar e configurar os *datapaths*. Quando uma modificação ocorre no ovsdb, o ovs-vswitchd é notificado e atualiza suas configurações.

O *datapath* que utiliza o módulo *kernel* “escuta” pacotes de uma interface de rede. No espaço do *kernel*, não é alocada muita memória, então se pode armazenar poucas regras de fluxo. Se o *datapath* não encontrar uma correspondência no *cache* da tabela de regras de fluxos no espaço do *kernel*, ele passa os pacotes para o ovs-vswitchd, que executa no espaço de usuário.

O ovs-vswitchd possui todas as entradas das regras de fluxo dos *datapaths*, que foram carregados da base de dados. Assim, o ovs-vswitchd atualiza a tabela de fluxos do espaço de *kernel* com a regra correspondente, e caso não tenha uma, realiza o procedimento de encaminhar os pacotes ao controlador e aguardar uma nova regra, para então atualizar a tabela no espaço *kernel*. A partir disso, o ovs-vswitchd reinjeta os pacotes no *datapath*. Por fim, o *datapath* encaminha ou descarta os pacotes com base nas entradas da tabela de fluxos em *cache* por uma interface de rede.

2.3.3.1 OVDSB

A interface de gerenciamento *Open Virtual Switch Data Base* (OVSDB) (traduzido como base de dados do *switch* virtual de código aberto) é usada para realizar o gerenciamento e configuração das operações na instância OVS (*switch* virtual) (PFAFF, 2013). O OVSDB não executa operações por-fluxo, este trabalho é do OpenFlow, no entanto outras operações são suportadas:

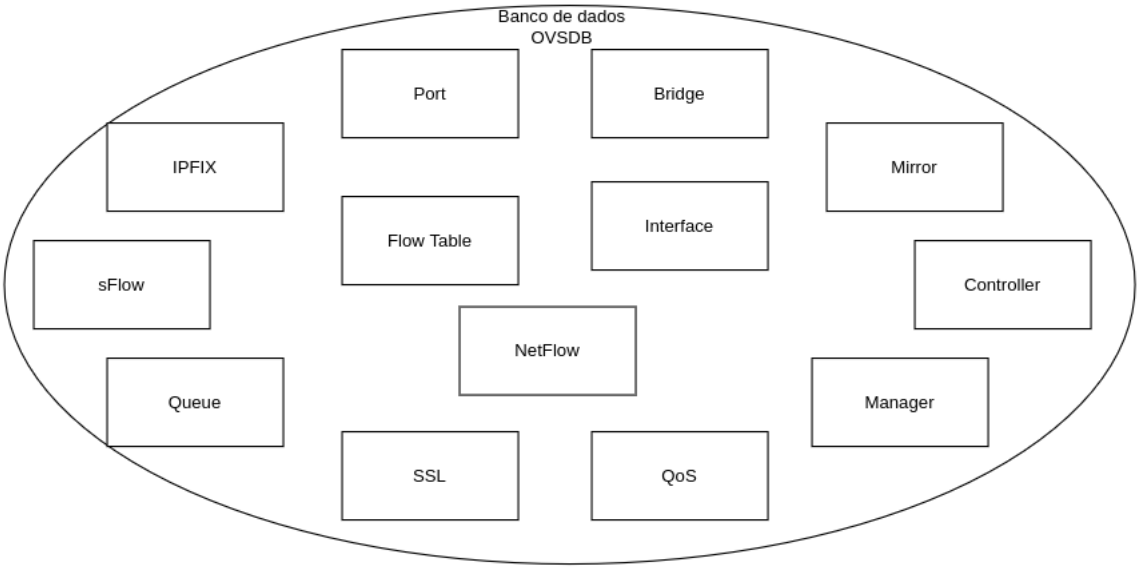
- Criação, modificação e exclusão de *datapaths* (*brigdes*) e interfaces virtuais de uma instância OVS.
- Configuração do conjunto de controladores ao qual um *datapath* deve se comunicar.
- Criação, modificação e exclusão de portas nos *datapaths*.
- Criação, modificação e exclusão de filas.
- Criação de políticas de QoS e vinculação às filas.
- Coleta de estatísticas de rede, relacionadas às *bridges* .

A Figura 10 apresenta o banco de dados do OVSDB contendo 13 tabelas, que armazenam as configurações utilizadas no OpenFlow (MAKAM, 2014). Um servidor OVSDB armazena informações de diversos *switches* virtuais em um mesmo banco de dados. Desta forma, as operações OpenFlow implementadas nos *switches* (como criação de regras) são registradas em uma tabela específica para cada operação.

2.3.3.2 Tabelas OpenFlow

As tabelas OpenFlow armazenam as regras de encaminhamento de fluxos para um *datapath*. Conforme explica (Selvaraj; Nagarajan, 2017), quando um pacote chega em um *switch*,

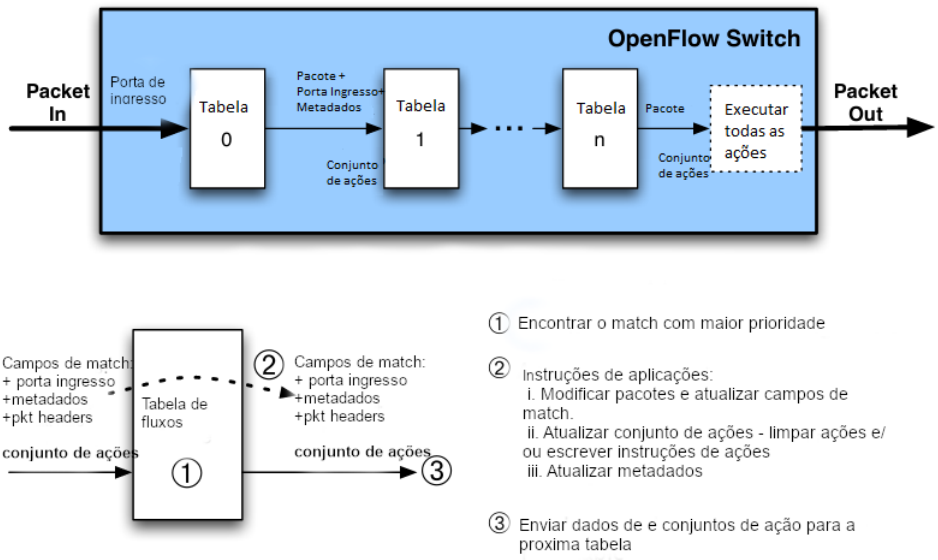
Figura 10 – Tabelas do banco de dados OVSDB.



Fonte: (Próprio Autor, 2022).

os dados do cabeçalho são analisados para encontrar uma correspondência nas regras das tabelas de fluxo instaladas. A Figura 11 apresenta esse processo. Assim, em geral, os *switches* são capazes de realizar três ações básicas, podendo encaminhar o pacote, descartar ou enviar para o controlador. Entretanto, também é possível trabalhar com ações mais sofisticadas, como alteração de campos de cabeçalho dos pacotes neste processo.

Figura 11 – Fluxo de funcionamento das tabelas.

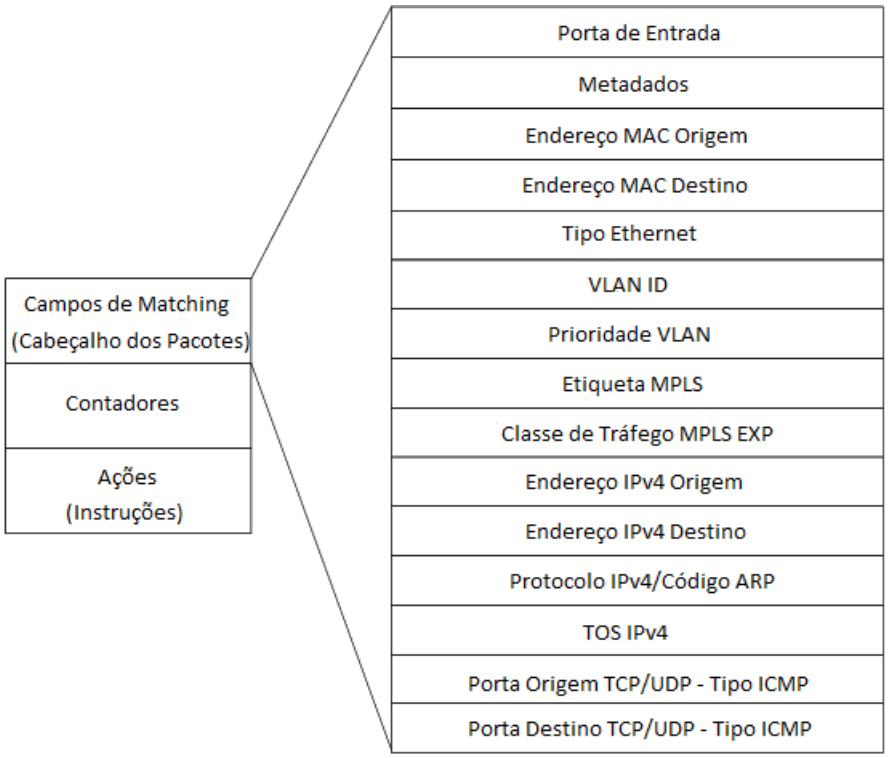


Fonte: Adaptado de ((ONF), 2012).

As tabelas de fluxos são conjuntos de instruções para fluxos e são armazenadas em um banco de dados específico (OVSDB). As regras de fluxo instaladas nas tabelas podem utilizar diversos campos para encontrar os pacotes alvo das ações definidas. A Figura 12 representa que

uma regra de fluxo é composta de campos de combinação, contadores e instruções.

Figura 12 – Regras de fluxo das tabelas OpenFlow.



Fonte: Adaptado de (BHOLEBAWA; DALAL, 2016).

Os campos de combinação também são apresentados na Figura 12. Já os contadores são divididos em: contadores para tabelas de fluxo, para entrada de fluxo, por porta, por fila, por grupo e por *meter*. Por fim, as instruções são conjuntos de ações associadas às regras de fluxos, que são executadas quando um pacote encontra uma combinação. Os conjuntos de instruções suportam várias ações, entre elas:

- Definir a porta de saída do pacote.
- Definir a fila associada a porta de saída, na qual o pacote será inserido.
- Descartar o pacote.
- Adicionar ou remover *tags*, como *tags Virtual Local Area Network* (VLAN) ou MPLS.
- Associar um grupo, assim o pacote é processado através das regras daquele grupo.

As regras de fluxo podem ser configuradas para definir em qual fila de uma porta de saída os pacotes desse fluxo devem ser encaminhados. Isso permite utilizar das filas de prioridade do HTB para diferenciar tráfego com priorização. Além disso, é possível utilizar uma lógica de múltiplas tabelas e realizar o encaminhamento de pacotes entre as regras dessas tabelas até uma regra de encaminhamento final. Com isso, as regras podem agir com efeito de cadeia e múltiplas

ações podem ser executadas sobre um mesmo fluxo de dados (conjunto de pacotes que satisfaz uma regra de fluxo).

2.3.3.3 Meter Table

Uma tabela *meter* consiste em medidores associados a fluxos (ONF, 2012). A Figura 13 apresenta os campos de uma tabela *meter*. Esses medidores medem um tipo de taxa nos pacotes, que é especificado na sua definição, e permitem controlar o fluxo de pacotes conforme a taxa definida. Este componente permite implementar diversas operações de QoS, como limitar taxas de transferências dos fluxos e podem ser combinados com estratégias de enfileiramento nas portas de saída dos *switches*.

Os medidores são associados com as ações das regras de fluxos. Desta forma, múltiplos medidores podem ser aplicados em um mesmo fluxo, por meio de tabelas de fluxo sucessivas. Cada entrada da tabela *meter* é identificado por seu identificador de medidor e contém os seguintes campos:

- *meter identifier* (identificador do *meter*): Inteiro sem sinal de 32 *bits* que identifica exclusivamente o medidor.
- *meter bands* (bandas do medidor): Lista de medidores, que medem algum componente específico e a forma de tratamento ao processar um pacote.
- *counters* (contadores): Atualizam quando os pacotes são processados por um medidor.

Figura 13 – *Meter table*.

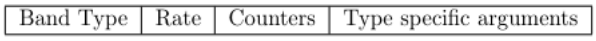
Fonte: Adaptado de (ONF, 2012).

Cada regra de *meter band* contém:

- *band type* (tipo de banda): Define como os pacotes são processados.
- rate (taxa): Define a taxa limite de encaminhamento que deve ser aplicada ao fluxo.
- *counters* (contadores): São atualizados quando um pacote é processado.
- *type specific arguments* (argumentos de tipo específicos): Alguns tipos de bandas suportam argumentos opcionais, como descarte pacotes se o fluxo passar da taxa definida.

A Figura 14 apresenta um desses medidores (*meter bands*), que serve para medir a taxa de entrega máxima de um fluxo de pacotes (ONF, 2012). Assim, fluxos que excedem o determinado pela regra sofrem reajuste por meio de descarte de pacotes.

Figura 14 – *Meter bands*.



Fonte: Adaptado de (ONF, 2012).

Portanto, as regras da *meter table* são fundamentais em mecanismos de gerenciamento de recursos de rede, pois permitem o *shapping* dos fluxos a taxas desejadas. Aliando esse componente com as filas de prioridade HTB e os modelos de alocação de largura de banda, é possível atingir os principais requisitos de tráfego de dados de aplicações de cidades inteligentes: QoS e priorização.

2.4 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram abordados os principais componentes que envolvem os cenários de cidades inteligentes e o tratamento de QoS de forma dinâmica por meio de redes *SDN*. Primeiramente, foram estudados os conceitos de cidades inteligentes e a necessidade de se implementar mecanismos para melhorar a qualidade de aplicações de cidades inteligentes, junto com o crescimento sustentável e eficiente das cidades.

Foram apresentadas algumas das diversas arquiteturas existentes para cidades inteligentes, que implementam o uso de dispositivos IoT e não-IoT para desenvolver soluções e aplicações para as cidades. Com isso, foi possível compreender os principais aspectos que envolvem o desenvolvimento de uma cidade inteligente.

A partir disso, foram apresentados os meios de geração e coleta de dados de cidades inteligentes, apresentado diversos tipos de dispositivos que podem ser utilizados pelas plataformas. Além disso, foi explicada a importância dos dados coletados e a forma com que eles são encaminhados na rede até chegarem em um serviço de nuvem para processamento.

Os dispositivos alimentam as aplicações das plataformas. Por isso, também foram explicados os tipos de aplicações que são desenvolvidas para cidades inteligentes, como sendo principalmente aplicações relacionadas à: negócios, cidadãos, ambiente físico e administração das cidades.

Assim, para entender os requisitos que os fluxos de dados dessas aplicações necessitam para trafegar com QoS, foram apresentados os principais tópicos que norteiam o gerenciamento de largura de banda e priorização em dispositivos de rede dentro de um sistema autônomo. Os estudos dos requisitos de aplicações apresentados na forma da Tabela 1 ajudam a compreender o quanto de cada recurso as aplicações necessitam e ajudam a modelar estratégias de gerenciamento baseados nessa informação. Com isso, foram explicados alguns métodos utilizados para emissores acordarem os valores dos requisitos necessários para seus fluxos de dados com os provedores de rede.

Os mecanismos para identificação desses fluxos foram apresentados para explicar como se pode tratar fluxos individualmente na rede. Além disso, foi explicada a dificuldade de implemen-

tar as estratégias de gerenciamento e fornecimento de recursos de rede em redes convencionais e que por isso, as redes SDN tem maior vantagem na implementação de estratégias de gerenciamento dinâmico de rede.

Assim, foram explicados dois modelos de gerenciamento de recursos de rede e priorização, que são aplicados no *framework* proposto. O método G-BAM propõe a divisão da largura de banda em classes de serviços que compartilham recursos de forma oportunista. Enquanto que o modelo de priorização utiliza a funcionalidade das filas HTB, onde cada fila pode possuir uma prioridade que interfere na velocidade de desenfileiramento. Assim, fluxos com maior prioridade devem ser encaminhados para filas de maior prioridade, enquanto fluxos de menor prioridade devem ser encaminhados para filas de menor prioridade.

3 TRABALHOS RELACIONADOS

A arquitetura SDN tem sido muito explorada em aplicações de fornecimento de QoS e disso surgem diversas estratégias de gerenciamento de rede que envolvem a programação dos dispositivos de rede e a distribuição de seus recursos entre fluxos de dados. Em grande parte dos trabalhos se opta por utilizar *switches* virtuais, como o OVS, para redução de custos e facilidade de implementação. Esse *switch* suporta o protocolo de comunicação OpenFlow, que é utilizado na comunicação com os controladores.

O trabalho de (TORRES et al., 2020) visa propor um *framework* chamado *Bandwidth Allocation Model through Software Defined Networking* (BAMSDN), para suprir os requisitos heterogêneos de usuários e aplicativos na alocação de recursos de forma dinâmica em redes MPLS SDN. O *framework* implementa um modelo BAM dividindo a banda em três classes de tráfego. Assim, os interessados em enviar dados pela rede realizam solicitações *Language Service Provider* (LSP) determinando uma quantidade de banda e a classe, de modo que o modelo BAM aloca os recursos para esses fluxos ao longo da rota (caminhos etiquetados MPLS). No entanto, esta abordagem não trata os requisitos individuais dos fluxos, pois os divide em classes, que são bem mais abrangentes e abstratas. Além disso, o modelo foi desenvolvido para funcionar em um domínio apenas, o que não permite ser utilizado em redes grandes.

O trabalho de (LEONARDI; BELLO; AGLIANò, 2020), propõe um sistema de admissão de fluxos e um gerenciador de recursos de rede. No modelo de divisão de recursos, a largura de banda é separada em níveis de prioridade, sendo elas: baixa, alta e alarme. No caso, são reservada partições fixas para cada classe, no entanto a classe mais baixa utiliza quanto estiver disponível. O mecanismo de admissão escuta requisições de nós finais, que informam a prioridade e a largura de banda solicitada. Deste modo, o controlador cria as regras controlando o uso da largura de banda por não alocar mais recursos do que o suportado pela classe.

O trabalho de (JANG; LIN, 2017) apresenta um *framework* para gerenciamento de largura de banda para *Internet service provider* (ISP)s SDN. Esses ISPs administrariam diversos dispositivos IoT de *smart homes*. No modelo proposto, os dispositivos são divididos entre dispositivos *Machine-to-Machine* (M2M) (sensores e afins) e dispositivos non-M2M (celulares, computadores e afins), bem como os serviços que podem ser acessados com esses dispositivos são divididos em oito categorias de QoS *Class Identifier* (QCI), cada um com características de QoS diferentes. Desta forma, o ISP estima um peso para cada categoria e divide a largura de banda conforme essas classes.

Uma grande necessidade das aplicações de rede é receber os recursos necessários para trafegar dados com qualidade e os trabalhos relacionados apresentam suas soluções para o problema. HiQoS é um *framework* proposto em (YAN et al., 2015), que fornece QoS em fluxos classificados como vídeo, vídeo/áudio interativo e *best-effort*. Essa abordagem cria uma fila para cada fluxo e limita a largura de banda pela configuração da fila. Além disso, utiliza um mecanismo de escolha de melhor caminho, para evitar congestionamento e entregar pacotes

de fluxos interativos com menor atraso. Em (TOMOVIC; PRASAD; RADUSINOVIC, 2014), também é apresentado um *framework* para reserva de largura de banda, onde obtém os requisitos a partir de um arquivo, e limita o consumo por meio de criação de filas individuais. Essa aplicação também monitora o uso da largura de banda e calcula rotas baseado na utilização.

O trabalho apresentado em (AGLIANò et al., 2018), desenvolve um sistema de gerenciamento de recursos para ambientes multi-controlador. Os controladores consultam a mesma base de dados para conhecer os requisitos dos recursos para os fluxos, que precisam ser pré-configurados. A utilização dos enlaces passa pela consulta dos *switches*, ao invés de salvar os estados no *framework*. No entanto, não é explicado como a largura de banda é reservada e gerenciada de fato.

O trabalho de (EGILMEZ; TEKALP, 2014) apresenta três contribuições: uma topologia com métodos de adquirir informações do estado de rede, um *framework* para fornecer QoS fim-a-fim em redes multi-domínios e um modelo de plano de controle distribuído para comunicação entre controladores. Neste modelo, a rede é particionada em domínios, onde cada domínio é gerenciado por um controlador. Assim, com o *framework* proposto, cada controlador realiza o gerenciamento de roteamento no seu domínio e troca informações sobre o estado das rotas com outros controladores, para identificar quais rotas garantem menor atraso. No entanto, as estratégias propostas se limitam a gerenciar rotas e os controladores estão permanentemente conectados, mas não explica como.

FlowQoS é um sistema proposto em (SEDDIKI et al., 2015) que executa sobre um *gateway* em uma rede doméstica para se comunicar com um controlador e prover QoS. Nesse sistema, os clientes definem os requisitos das aplicações por meio de uma página WEB. A partir dos dados dessa página o controlador da rede identifica o tipo de aplicação e seus requisitos de QoS, para criar as regras de encaminhamento nos dispositivos de rede. A arquitetura proposta utiliza de duas camadas de OVS entre origem e destino. Com dois *switches* simulando um só, é criado um *link* entre os dois para cada classe de serviço, com as limitações de largura de banda de cada classe. Então, para cada fluxo admitido é criado uma regra limitadora de tráfego utilizando *tc*.

Em (DIORIO; TIMóTEO, 2016) é proposto um mecanismo de fornecimento de recursos para fluxos de aplicações multimídia utilizando SDN. Neste modelo, um *gateway* se conecta com os *hosts* que consomem fluxos multimídia, para identificar e classificar os tipos de tráfego em requisitos de largura de banda e prioridade. Esse *gateway* serve como um complemento ao controlador SDN, de modo que o *gateway* é responsável por gerenciar a largura de banda e o encaminhamento de pacotes, depois, apenas informa as decisões para o controlador SDN por meio de uma API RESTful (*Northbound*).

Em (KRISHNA; ADRICHEM; KUIPERS, 2016), foi proposto um método de fornecimento de QoS fim-a-fim utilizando SDN para promover reserva de recursos para fluxos individuais ao longo do caminho da origem até o destino. Para isso, é utilizado um algoritmo de roteamento chamado *Widest-Shortest Path* juntamente com um grafo que representa a rede e

informações sobre as reservas de largura de banda entre os *links*. A estrategia utilizou a implementação de uma fila HTB por fluxo nos *switches* para promover o *shapping* na largura de banda. Além disso, por serem filas HTB, é permitido o compartilhamento de banda não utilizada.

A solução abordada em (COSTA; OLIVEIRA, 2020) leva em consideração otimizar o tráfego de aplicações em *smart cities* por meio da identificação da prioridade desses fluxos. Este trabalho assume que os eventos de uma cidade, como incêndios e acidentes, influenciam na importância que os dados coletados na cidade possuem para as aplicações que os utilizam. Por isso, os fluxos de dados devem ser diferenciados em termos de prioridade, para que sejam tratados com mais recursos e agilidade na rede. Para isso, um nó servidor, referido como módulo de processamento central (*gateway* IoT) armazena e distribui para os dispositivos, uma tabela de eventos e uma tabela de contexto, que possuem valores pré-estabelecidos. A tabela de eventos associa um evento à uma prioridade, já a tabela de contexto, associa eventos que não possuem ligação direta com a aplicação, mas que podem influenciar no seu funcionamento. Desta forma, cada dispositivo possui sua tabela e calcula sua prioridade individualmente, permitindo que sejam tratados de forma diferenciada na rede. Esta abordagem é interessante pois traz a possibilidade de dispositivos de cidades inteligentes identificarem a sua prioridade, por meio do uso de um *gateway*, que permite que fluxos de diferentes prioridades possam ser tratados de forma dinâmica em redes que suportem programabilidade.

Os trabalhos de (AGLIANò et al., 2018), (SEDDIKI et al., 2015) e (EGILMEZ; TEKALP, 2014) são os que fornecem soluções multi-domínio, com reserva de largura de banda conforme os requisitos dos fluxos por criação de filas de prioridades específicas. No entanto, a sincronização entre controladores envolve ferramentas terceiras e o gerenciamento de uso de banda são por criação de filas.

Os trabalhos explorados utilizam formas de se obter os requisitos dos fluxos são por monitoramento ou pré-configuração, na maioria. As aplicações que não são para cenários multi-controladores são limitadas a redes pequenas para fornecer QoS fim-a-fim. A Tabela 6 comprime os principais pontos dos trabalhos relacionados. As garantias de QoS por reserva de recursos são geralmente por meio de criação de filas HTB com limite de transferência, ao invés de usar regras meter do protocolo OpenFlow. Além disso, as soluções preferem atualizam os estados de uso de banda pelas classes de serviços, quando definidas, fazendo consultas aos *switches* ou monitorando os enlaces, o que é bem mais custoso que a solução proposta, que salva os fluxos no *framework*.

Tabela 6 – Comparativo entre trabalhos relacionados.

Referência	Multi-controlador	Descoberta de QoS	Reserva de recursos	Divisão de largura de banda
(Mocelin Jr., 2022)	Sim	Contrato	Regras meter + HTB	8 filas de classes com prioridade (HTB) + G-BAM
(AGLIANò et al., 2018)	Sim	Base de dados remota	Não identificado	Não identificado
(SEDDIKI et al., 2015)	Não	Página WEB	Duas camadas OVS + TC	Classes de serviços
(EGILMEZ; TEKALP, 2014)	Sim	Um interface para emissores - controladores	Não	Classes de serviços
(KRISHNA; ADRICHEM; KUIPERS, 2016)	Não	Baseado em IP	Fila HTB por fluxo	Classes de serviço com prioridade
(DIORIO; TIMóTEO, 2016)	Não	Gateway marca DSCP e controlador identifica	Uma fila por classe (best-effort)	Classes de serviços
(YAN et al., 2015)	Não	Baseado em IP	Fila HTB por fluxo	Classes de serviços
(TOMOVIC; PRASAD; RADUSINOVIC, 2014)	Não	Um arquivo pré-configurado	Fila HTB por fluxo	Classes de serviços
(JANG; LIN, 2017)	Não	Um mecanismo identificador de classes	Não informa	Classes de serviços e priorização
(LEONARDI; BELLO; AGLIANò, 2020)	Não	Primeiro pacote de um fluxo contém seus requisitos (packet_in)	Reserva por classe (best-effort)	Classes de prioridade
(TORRES et al., 2020)	Não	Baseado em IP (pré-configurado)	Fila para cada classe (best-effort)	Classes de serviços

Fonte: (Próprio Autor, 2022).

3.1 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram apresentados os trabalhos relacionado com a abordagem proposta. Nos trabalhos, é possível observar a preferência por desenvolver soluções de fornecimento de QoS voltados para arquiteturas de redes SDN, devido a flexibilidade de se programar os dispositivos de redes de modo centralizado, por meio do controlador. A maioria das soluções propostas visam separar a largura de banda dos enlaces em classes de serviços e fornecer reserva de largura de banda baseado nessa divisão. No entanto, não foram encontrados trabalhos que se aproveitassem do mecanismo de limitação de largura de banda fornecido pelo OpenFlow, que são as regras *meter*. Deste modo, o *shaping* dos fluxos e o controle do uso de largura de banda nos enlaces precisa ser realizado por outras técnicas, que muitas vezes são difíceis de implementar para soluções dinâmicas, como é o caso da limitação de largura de banda utilizando filas *HTB* por fluxo. Além disso, diferentemente do trabalho proposto, as soluções não armazenam os estados dos enlaces no controlador e, grande parte, não é voltada para ambientes distribuídos multi-domínios.

4 ABORDAGEM PROPOSTA

Nesse capítulo é apresentado o *framework* proposto bem como sua implementação. Este trabalho visa desenvolver uma solução para contribuir com as áreas de qualidade de serviço e de aplicação de cidades inteligentes. O FLOWPRI-SDN é um *framework* para gerenciamento e alocação de largura de banda de forma priorizada e distribuída, que pode ser aplicado em cenários de cidades inteligentes para suprir os requisitos de comunicação de suas aplicações de forma dinâmica. Este *framework* é uma aplicação construída sob o controlador Ryu e atua no gerenciamento dos dispositivos de dados de um domínio SDN.

Neste trabalho, a cidade inteligente é representada por dispositivos IoT e não IoT (*smartphones* e computadores) que se localizam na borda da rede (cidade) e um serviço de nuvem, que representa a central de processamento das aplicações de cidades inteligentes. Estes componentes se conectam por uma rede SDN, que é dividida em diversos domínios. Cada domínio possui seus *switches* para encaminhamento de dados e suas políticas de escolha de rotas. Assim, os domínios utilizam dessas rotas em conjunto com as políticas implementadas pelo *framework* para controlar os *switches* e fornecer alocação de largura de banda gerenciada para os fluxos de dados admitidos.

Na cidade inteligente, os dispositivos IoT podem estar conectados em redes próprias (que utilizam os protocolos IoT das camadas de rede e aplicação) em determinados locais da cidade, onde acessam a rede da cidade por meio de um *sink node*. Ou então, podem acessar a rede por conta própria, por uma conexão direta com um *switch* de borda, como é o caso explorado no cenário de teste. Por outro lado, os dispositivos não-IoT possuem recursos de computação e comunicação suficientes para se conectarem a um *switch* da rede por conta própria.

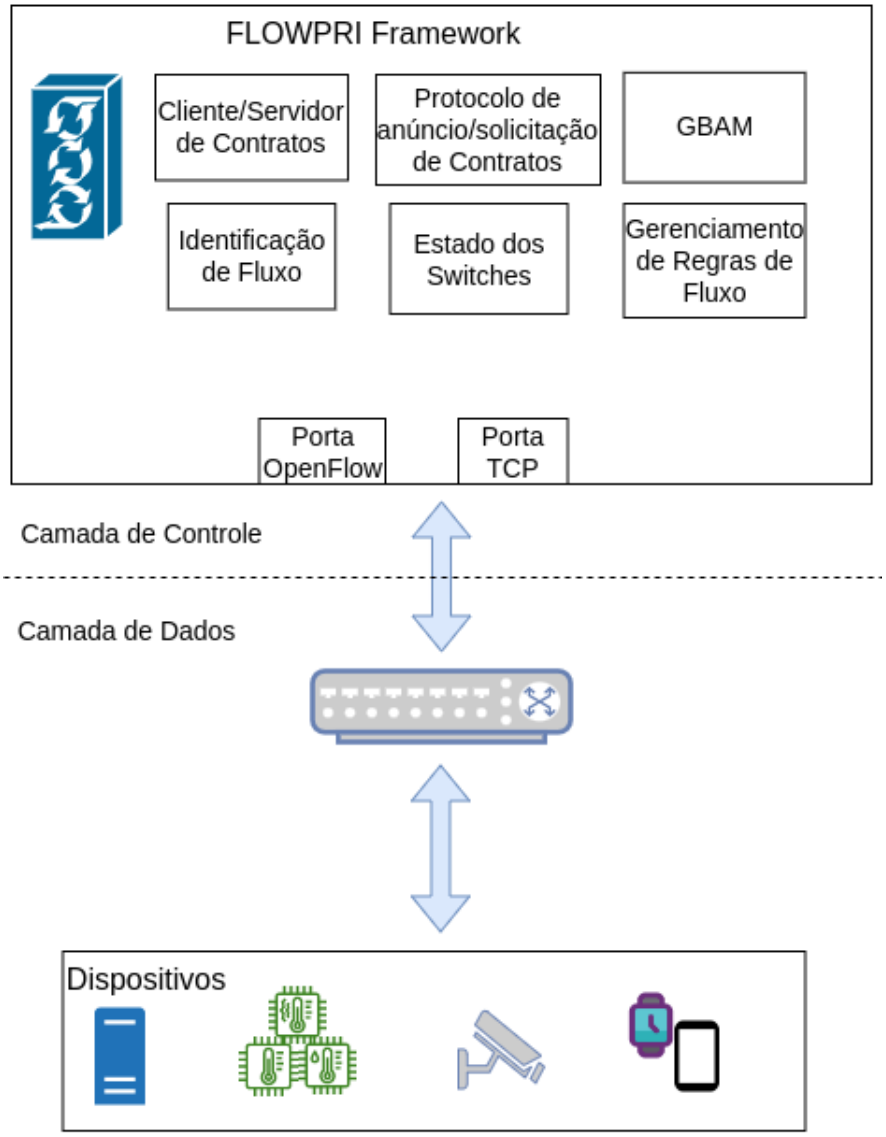
Esses dispositivos podem acessar aplicações de cidades inteligentes e também acessar serviços que não possuem relação com a cidade inteligente. Neste caso, o tráfego passa a ser visto como de *background*. Esse tipo de tráfego representa o tráfego que existiria em um cenário em que as cidades inteligentes estivessem conectadas pela Internet e deve ter menor prioridade que o tráfego dos serviços voltados para a cidade inteligente.

4.1 ARQUITETURA DO FLOWPRI-SDN

Os elementos internos do *framework* FLOWPRI-SDN são apresentados nessa seção. Conforme a arquitetura apresentada na Figura 15. O FLOWPRI-SDN é construído sob o controlador SDN Ryu de modo a gerenciar os *switches* de seu domínio e interagir com dispositivos emissores de dados da rede. Para orquestrar os *switches* do domínio, o *framework* utiliza mensagens do protocolo OpenFlow, que são enviadas por uma porta restrita conectada por um salto com cada *switch* OVS. Por outro lado, para se comunicar com outros dispositivos e controladores, o FLOWPRI-SDN utiliza uma porta TCP comum, que torna o *framework* acessível pelo plano de dados SDN. As principais ações realizadas pelo *framework* são: Gerenciar os estados dos *switches*, identificar fluxos, alocar fluxos conforme a política de reserva priorizada de recursos,

enviar e receber contratos de QoS. As funções que permitem identificar essas capacidades são descritas na seção 4.2.

Figura 15 – Arquitetura FLOWPRI-SDN



Fonte: (Próprio Autor, 2022).

O FLOWPRI-SDN implementa a modelagem de divisão de largura de banda para portas físicas dos *switches* conforme apresentada em (CISCO, 2008), por meio de filas HTB. Entretanto, esse tipo de configuração precisa ser realizada "a mão"usando a ferramenta do OpenFlow "ovs-vsctl", devido a versão OpenFlow utilizada (2.13.1) não possuir mensagens para o propósito de configuração de filas nas interfaces, apenas para obter seus estados.

A empresa Cisco apresenta uma configuração para fornecimento de QoS para fluxos utilizando filas de classes de serviços, de modo que aplicações de áudio, vídeo e dados coexistam na rede com os requisitos apropriados de QoS garantidos (CISCO, 2008). A recomendação é que não seja reservado de forma fixa, mais que 33% da largura de banda disponível para fluxos de tempo-real, 25% para fluxos que não possuam requisitos especificados (*best-effort*), 7% para

tráfego de controle (como *Internet Control Message Protocol* (ICMP)) e os 35% restantes para fluxos que não sejam de tempo-real.

O FLOWPRI-SDN implementa essa estratégia de divisão de largura de banda junto com o conceito de filas de prioridade por meio de filas *HTB*. Para o propósito de priorização, foi definido utilizar três níveis de prioridade para cada classe de serviço que o *framework* implementa tratamento de reserva de recursos, sendo as classe de fluxos de tempo-real e de fluxos não-tempo-real, e um nível de prioridade para classes que não são implementadas com reserva de recursos, sendo as classe de fluxos de controle e fluxos *best-effort*.

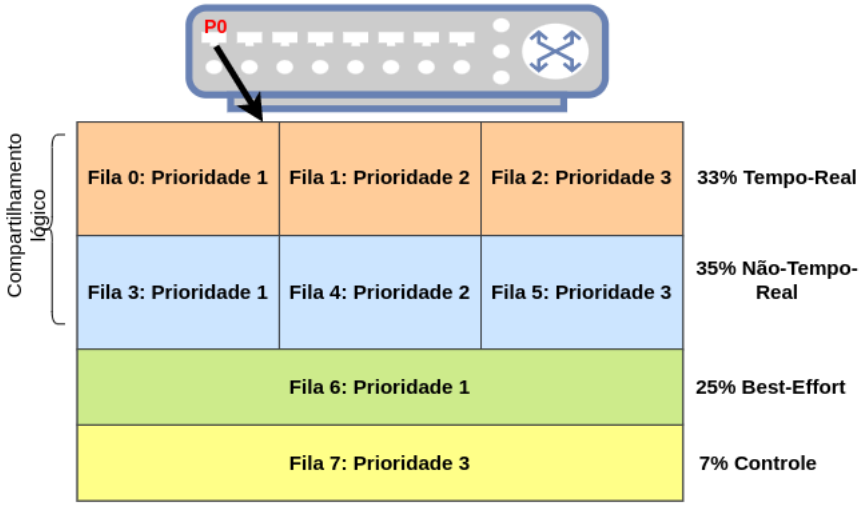
Como o HTB não é completamente suportado pelo OVS, só é possível criar uma camada de hierarquia de filas. Por isso, as classes de tempo-real e não-tempo-real são representadas por três filas cada: prioridade 1 (alta), prioridade 2 (media), prioridade 3 (baixa), conforme mostrado na Figura 16. As prioridades das filas de *best-effort* e controle, são baixa e alta respectivamente e para essas classes é criada uma fila para cada. O FLOWPRI-SDN armazena o estado de cada fila de prioridade dos *switches*, ficando responsável pelo controle do uso de largura de banda, para não permitir que a soma do uso da largura de banda das filas seja maior que a limitada para cada classe, a menos que existam fluxos emprestando banda de outra classe (no caso entre tempo-real e não-tempo-real).

Na abordagem proposta, os mecanismos de empréstimo de largura de banda ocorrem de duas formas: empréstimo com reserva de largura de banda e sem reserva de largura de banda. O empréstimo com reserva de largura de banda é implementado pelo modelo de alocação de largura de banda G-BAM juntamente com regras limitadoras de tráfego, as regras *meter*. Por outro lado, o empréstimo sem reserva de largura de banda é implementado utilizado o mecanismo de distribuição de banda não utilizada do *HTB*. O modelo G-BAM é responsável por orquestrar a alocação de largura de banda entre os fluxos e realizar o empréstimo de banda entre as classes tempo-real e não-tempo-real com reserva de recursos garantida, caso o fluxo seja aceito. Por sua vez, o modelo de filas HTB, é utilizado para realizar o empréstimo da banda não utilizada por todas as filas das classes, sem reserva de recursos, para a fila da classe *best-effort*. A fila de tráfego de controle de rede não empresta largura de banda de outras classes.

Dessa forma, os *switches* configurados com o modelo de filas apresentado, possuem oito filas em cada porta de saída para encaminhar o tráfego egresso conforme seu perfil de classe e prioridade. Assim, para que os *switches* sejam capazes de identificar os fluxos e separá-los corretamente, são definidos códigos DSCP específicos, baseados nos estudos de requisitos de tráfego de (JANEVSKI; JANKOVIC; MARKUS, 2017). Um código DSCP serve para identificar a classe, a prioridade e a largura de banda de um fluxo de dados. Além disso, os códigos DSCP servem para limitar os perfis de largura de banda que o *framework* suporta e evitar que fluxos solicitem largura de banda sem controle.

Assim, baseado nas informações dos fluxo de dados acordadas entre dispositivos emissores e o *framework*, em termos de sua classe de serviço, largura de banda e prioridade, é procurado o código DSCP que supre a necessidade do fluxo. O mecanismo de acordo de QoS

Figura 16 – Configuração das portas dos *switches*.



Fonte: (Próprio Autor, 2022).

que informa os requisitos de QoS dos fluxos é explicado na subseção 4.2.3. A partir do código *DSCP* calculado de um contrato de QoS, o mecanismo de alocação de largura de banda G-BAM cria as regras de classificação e encaminhamento nas tabelas de fluxo dos *switches* do domínio. A regra de classificação marca o código *DSCP* correspondente, no campo DSCP do cabeçalho dos pacotes IPv4 do fluxo definido no acordo de QoS. Deste modo, a regra de encaminhamento é criada com *match* nos endereços IPv4 e no campo *DSCP*, onde o código desse campo identifica por qual fila da porta de saída os pacotes desse fluxo devem ser enviados.

Os valores DSCP suportados na versão implementada do FLOWPRI-SDN são apresentados nas Tabelas 7 e 8, incluindo também o DSCP 60 para fluxos *best-effort*, que são encaminhados pela fila 6 e o DSCP 61 para fluxos de controle, que são encaminhados pela fila 7.

Códigos DSCP podem ser escolhidos a partir de um conjunto de valores definidos no RFC 2474 ou podem ter significado puramente local (NICHOLS et al., 1998), que é o caso. O uso dos códigos não é obrigatório e as marcações de pacotes não podem ser assumidas como verdadeiras por todo o caminho, cada domínio é responsável pela marcação e controle dos códigos em seu espaço de controle. Por esse motivo e pelos códigos DSCP padrões não serem pensados em tratamentos de QoS para fluxos individuais, que o *framework* utiliza de seus próprios códigos e métodos de marcação.

Deste modo, como cada marcação DSCP tem validade apenas em seu domínio, é necessário que outros *frameworks* ao longo da rota dos fluxos tenham acessoa aos contratos de QoS e consigam calcular o código DSCP para os fluxos e fornecer as políticas de QoS solicitadas. Assim, o FLOWPRI-SDN foi desenvolvido para ser compatível com cenários multi-controladores. O protocolo de anúncio/solicitação de contratos desenvolvido faz uso de pacotes ICMP *Information Request* (tipo 15) e ICMP *Information Reply* (tipo 16) para realizar o envio dos contratos de QoS, entre domínios cujos controladores utilizem o FLOWPRI-SDN.

Esses códigos ICMP foram propostos no RFC 792 (POSTEL, 1981) com o propósito de

Tabela 7 – Códigos DSCP da classe de tráfego de tempo-real.

Tempo-Real			
Banda	Prioridade 1 DSCP:Fila	Prioridade 2 DSCP:Fila	Prioridade 3 DSCP:Fila
32kbps	1:0	11:1	21:2
64kbps	2:0	12:1	22:2
128kbps	3:0	13:1	23:2
500kbps	4:0	14:1	24:2
1Mbps	5:0	15:1	25:2
2Mbps	6:0	16:1	26:2
5Mbps	7:0	17:1	27:2
10Mbps	8:0	18:1	28:2
25Mbps	9:0	19:1	29:2

Fonte: (Próprio Autor, 2022).

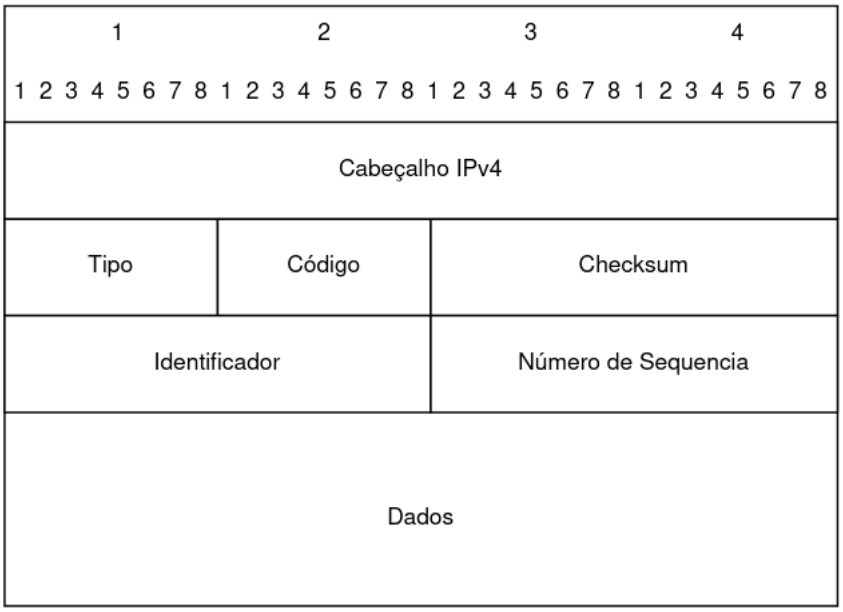
Tabela 8 – Códigos DSCP da classe de tráfego de fluxos que não são de tempo-real.

Não Tempo-Real			
Banda	Prioridade 1 DSCP:Fila	Prioridade 2 DSCP:Fila	Prioridade 3 DSCP:Fila
32kbps	31:3	41:4	51:5
64kbps	32:3	42:4	52:5
128kbps	33:3	43:4	53:5
500kbps	34:3	44:4	54:5
1Mbps	35:3	45:4	55:5
2Mbps	36:3	46:4	56:5
5Mbps	37:3	47:4	57:5
10Mbps	38:3	48:4	58:5
25Mbps	39:3	49:4	59:5

Fonte: (Próprio Autor, 2022).

um *host* descobrir o número da rede em que está e receber como resposta um endereço válido na rede. No entanto, este método era válido apenas quando a Internet estava em sua fase inicial e foi formalmente tornado obsoleto pelo RFC 6918 (GONT, 2013), com a justificativa de que novos mecanismos, como *Dynamic Host Configuration Protocol* (DHCP) (traduzido como protocolo de configuração dinâmica de *host*), substituíram esse tipo de mensagem, para fins de configuração de *hosts*. A Figura 17 destaca o cabeçalho do pacote ICMP de código 15 e 16. Esses pacotes específicos do protocolo de controle da Internet, apesar de terem sido descontinuados, servem para o propósito de anúncio e solicitação de contratos utilizados no FLOWPRI-SDN.

Figura 17 – Pacotes ICMP código 15 e 16.



Fonte: Adaptado de (POSTEL, 1981).

4.2 IMPLEMENTAÇÃO

Nessa seção é apresentada a implementação do FLOWPRI-SDN. O *framework* possui funcionalidades que aproveitam da comunicação com os *switches* e funcionalidades que aproveitam da comunicação com os dispositivos emissores e controladores. As funcionalidades que utilizam da comunicação entre *framework* e *switches* são ativadas por eventos OpenFlow e também de forma proativa. Mais especificamente, o FLOWPRI-SDN trata os eventos OpenFlow *switch_features*, *packet_in* e *flow_removed*, entretanto, pode proativamente criar e remover regras dos *switches* a qualquer instante. Por sua vez, as funcionalidade que envolvem a comunicação com dispositivos emissores e controladores são as de estabelecimento de contrato de QoS e do envio de mensagens do protocolo de anúncio/solicitação de contratos de QoS.

4.2.1 Configuração de Filas nas Portas dos *switches*

A configuração das filas HTB em cada porta dos *switches* é um requisito para o FLOWPRI-SDN, pois com elas se divide a largura de banda dos enlaces em classes de serviços. No entanto, por limitações do OpenFlow, do OVS e do Ryu, é necessário configurar "na mão". O *script* apresentado na Figura 18, mostra uma parte da configuração usada nos testes quando a largura de banda total de um enlace deve ser 10Mbps.

No *script*, nos primeiros seis comandos são definidas as variáveis com as larguras de banda de cada classe de serviço, facilitando a criação de várias filas com apenas um comando. Então, nos dois comandos seguintes, a configuração anterior da porta *s1-eth1* do *switch S*1 é removida. As configurações anteriores de largura de banda dos enlaces são removidas para que não entrem em conflito com as novas filas que são criadas. A partir disso, o próximo comando

Figura 18 – Script de criação de filas HTB.

```
BANDA=10000000
CLASS1=3300000
CLASS2=3500000
SOMACLASS12=6800000
CLASS3=2500000
CLASS4=700000

sudo ovs-vsctl clear port s1-eth1 qos
sudo tc qdisc del dev s1-eth1 root

sudo ovs-vsctl -- set port s1-eth1 qos=@newqos -- --id=@newqos create qos type=linux-htb other-config:max-rate=$BANDA
queues=0=@q0,1=@q1,2=@q2,3=@q3,4=@q4,5=@q5,6=@q6,7=@q7 --
--id=@q0 create queue other-config:min-rate=$SOMACLASS12 other-config:max-rate=$SOMACLASS12 other-config:priority=10 --
--id=@q1 create queue other-config:min-rate=$SOMACLASS12 other-config:max-rate=$SOMACLASS12 other-config:priority=5 --
--id=@q2 create queue other-config:min-rate=$SOMACLASS12 other-config:max-rate=$SOMACLASS12 other-config:priority=2 --
--id=@q3 create queue other-config:min-rate=$SOMACLASS12 other-config:max-rate=$SOMACLASS12 other-config:priority=10 --
--id=@q4 create queue other-config:min-rate=$SOMACLASS12 other-config:max-rate=$SOMACLASS12 other-config:priority=5 --
--id=@q5 create queue other-config:min-rate=$SOMACLASS12 other-config:max-rate=$SOMACLASS12 other-config:priority=2 --
--id=@q6 create queue other-config:min-rate=$CLASS3 other-config:max-rate=$BANDA other-config:priority=10 --
--id=@q7 create queue other-config:min-rate=$CLASS4 other-config:max-rate=$CLASS4 other-config:priority=2
```

Fonte: (Próprio Autor, 2022).

usa a ferramenta OVS *ovs-vsctl* para criar na porta *s1-eth1* do *switch S*1 as oito filas, com as divisões de largura de banda explicadas no inicio da seção 4.1.

As filas da classe 1 (tempo-real) são as de identificação 0 até 2 inclusive. As filas da classe 2 (não-tempo-real), são as de identificação 3 até 5 inclusive. Enquanto que as filas 6 e 7 são das classes 3 e 4, *best-effort* e controle, respectivamente. Além disso, cada fila de classe tem a sua prioridade (quanto menor o número mais prioritário), que determina a ordem para empréstimo de largura de banda e de retirada de pacotes em caso de enfileiramento. A informação *min-rate*, determina a largura de banda mínima garantida que a fila pode usar, sem que outra fila empreste largura de banda. Já o campo *max-rate*, determina até quanto de largura de banda uma fila pode emprestar das outras filas, quando esse recurso não estiver sendo consumido.

Cabe perceber que, as filas da classe 1 e 2 possuem a largura de banda mínima garantida como o somatório da banda total de suas classes. Isso se deve ao fato de que o FLOWPRI-SDN empresta de forma lógica largura de banda entre essas classes e com a garantia mínima, evita que outra fila utilize a largura de banda que seria emprestada. Já para o caso da fila da classe 3, ela pode usar toda a largura de banda disponível, menos a que está dentro do limite mínimo de cada uma das demais classes.

4.2.2 Gerenciamento dos Estados dos *Switches*

O FLOWPRI-SDN armazena as configurações dos *switches* OVS de seu domínio em forma de classes orientadas a objeto. Existe uma classe orientada a objetos para representar *switches*, uma para representar as portas dos *switches*, uma classe para representar as regras ativas dos *switches* e uma classe orientada a objetos para representar comandos OpenFlow a serem executados em um *switch*.

A Figura 19, descreve as relações entre as classes orientadas a objetos. A classe orientada a objetos principal representa o *framework* em si. Uma instância FLOWPRI-SDN pode possuir diversos *switches* em seu domínio, armazenados na forma de instâncias da classe *switch*,

juntamente com uma lista de contratos de QoS recebidos.

Um objeto *switch* possui um identificador único, uma lista de instâncias da classe porta e uma conexão com o *switch* OVS representado (*datapath*). Um objeto *switch* possui diversas funções para criar e remover regras da tabela de fluxos do *switch* OVS que representa.

Uma porta também é uma classe orientada a objetos, que representa uma porta de um *switch* OVS. Um objeto porta possui informações sobre a quantidade de largura de banda total para tráfego de classe de tempo-real, a quantidade de largura de banda total para tráfego da classe de fluxos não-tempo-real, a quantidade de banda consumida de cada classe e, para cada fila de prioridade de classe de serviço com QoS, uma lista de instâncias da classe regras de fluxo.

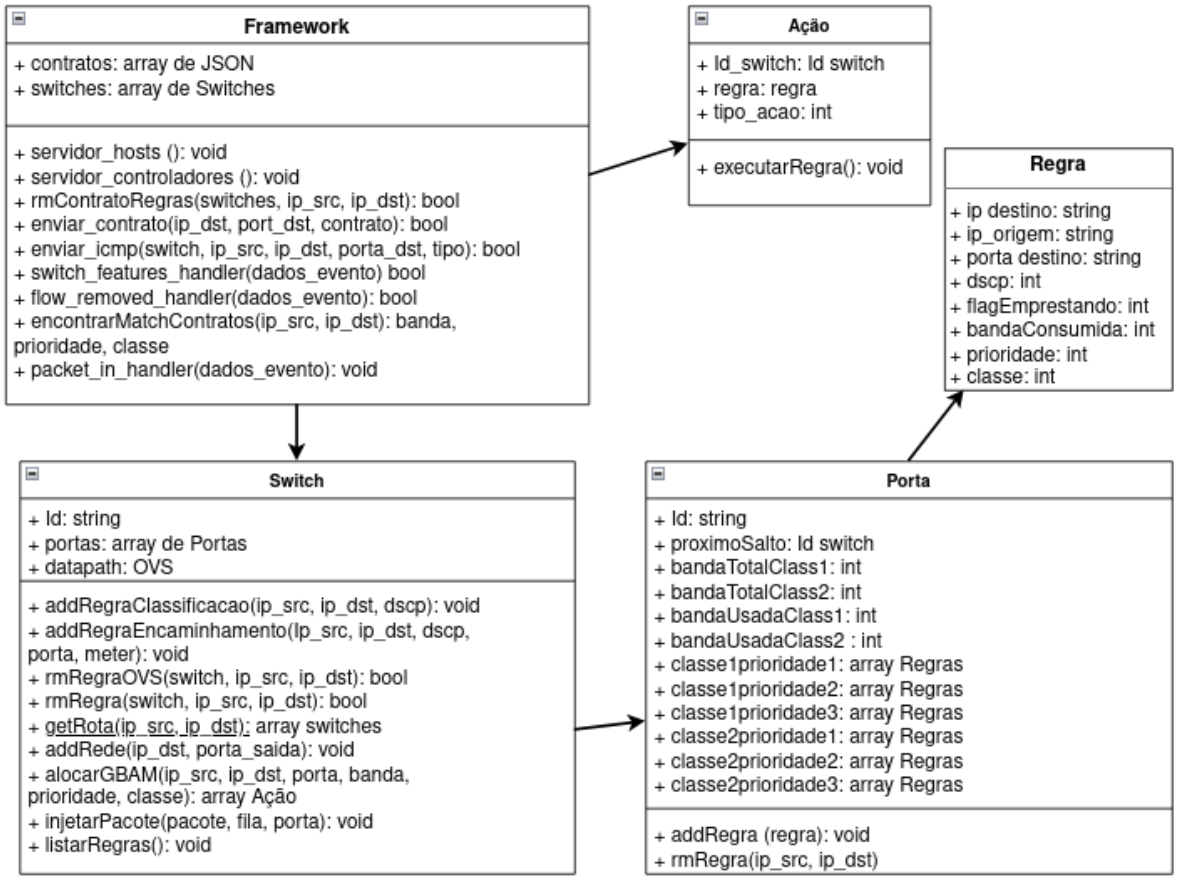
A classe orientada a objetos do tipo regra, é uma representação de regra de fluxo OpenFlow. Um objeto regra possui os campos para *matching*, que são os endereços IPv4 de origem e destino do fluxo e o código *DSCP*. Além disso, se tem os identificadores da fila de saída, que são os campos de prioridade e classe, o identificador de largura de banda consumida, o identificador que informa se essa regra está emprestando largura de banda e por fim, o identificador de porta de saída do fluxo. Deste modo, com base nessas informações, é possível representar qualquer regra OpenFlow que é criada pelo *framework*.

A classe orientada a objetos ação, define o que se deve fazer com uma regra criada pelo FLOWPRI-SDN nos processos de alocação de fluxos. Um objeto desse tipo, possui o identificador de em qual *switch* OVS essa ação deve ser executada e o campo de tipo de ação, que define se é uma ação de criação ou exclusão. Desta forma, baseado no tipo de ação, a função *executarRegra* chama a rotina de adição de regras no *switch* OVS ou chama a de remoção.

Deste modo a modelagem de *software* implementada habilita o FLOWPRI-SDN manter o estado dos *switches* OVS sem ter de consultá-los diretamente. Para isso, o primeiro evento tratado pelo FLOWPRI-SDN é o *switch_features*. Quando um *switch* OpenFlow inicializa e procura se conectar ao controlador, um pacote do tipo *switch_features* com os recursos do *switch* é encaminhado ao controlador.

Esse pacote gera o evento *switch_features*, que permite que os *switches* sejam configurados de forma proativa, por meio de definições preestabelecidas para cada *switch*. No FLOWPRI-SDN, esse evento é tratado para armazenar instâncias dos *switches* e configurá-los conforme o Algoritmo 1.

Figura 19 – Diagrama de Classes Armazenadas no FLOWPRI-SDN.



Fonte: (Próprio Autor, 2022).

Algorithm 1 TRATADOR DO EVENTO *switch_features*

Entrada: *switch_features*

início
- Obter o identificador do *switch* OVS que gerou o evento
- Carregar a configuração predefinida para o *switch* OVS;
- Obter o número de portas do *switch*;
- Obter a lista de identificadores das portas;
- Obter a lista de larguras de banda nominal de cada porta;
- Criar uma instância *switch* no *framework*;
- Armazenar a conexão para o *switch* OVS (*datapath*) na instância *switch*;
- Adicionar o identificador do *switch* OVS à instância *switch*;
- Adicionar as configurações das portas à instância *switch* (quantidade de portas, identificadores e larguras de banda);
- Armazenar a instância do *switch* na lista de *switches* no *framework*;
- Adicionar à instância do *switch*, os prefixos de rede e endereços IPv4 destino, junto com o identificador da porta de saída;
- **para** *cada porta da instância switch* **faça**
  - Adicionar o próximo salto, que é o identificador de outro *switch* do domínio, ou valor que identifica saída do domínio;
- **fim**
- Criar as regras padrão das tabelas de fluxo 0 e 1 do *switch* OVS usando o protocolo OpenFlow;

fim

Primeiramente, o *switch* OVS é identificado e então as configurações predefinidas para ele são carregadas. Com isso, são criadas as instâncias da classe *switch* e portas necessárias. Na sequência, em cada instância de porta são configuradas as rotas com os prefixos de destino ou endereços *IPv4*, conforme a topologia da rede pré-conhecida, bem como são criadas regras de fluxo padrão.

No FLOWPRI-SDN, os *switches* são configurados com duas tabelas de fluxo: tabela 0 - regras de classificação e tabela 1 - regras de encaminhamento. As regras de classificação são criadas apenas nos *switches* OVS mais próximos da borda emissora e após serem executadas, realizam alguma marcação no campo *DSCP* do cabeçalho IPv4 dos pacotes e ativam a ação de enviar para ser testado na tabela de encaminhamento. Por sua vez, as regras de encaminhamento são configuradas em cada tabela de fluxo dos *switch* OVS da rota e determinam qual a fila e porta de saída para encaminhar o pacote.

A primeira regra padrão da tabela 0 é para encaminhar pacotes que não possuem correspondência para a tabela 1. Já a segunda regra é criada na tabela 1, para encaminhar pacotes que não possuem correspondência para o controlador e gerar o evento *packet_in*. Além dessas duas regras, são criadas as regras para encaminhar os pacotes com destino ao endereço do *framework*. Neste caso, na tabela 0 são criadas regras para que pacotes com destino ao *framework* sejam marcadas com DSCP de controle e encaminhados para a tabela 1, na qual é criada uma regra para que esse fluxo seja encaminhado pela fila de controle (identificação 7) para a interface que leva ao controlador, conforme a rota definida para isso.

Após a execução da função que trata o evento *switch_features*, os *switches* possuem regras padrão instaladas semelhantes a da Figura 20. As regras das tabelas de fluxo dos *switches* OVS podem ser listadas a qualquer momento pelo comando "ovs-ofctl -O OpenFlow –stats -m dump-flows <switch>".

Figura 20 – Regras padrão instaladas no *switch* OVS *S*1 após o evento *switch_features*.

```
 cookie=0x0, duration=130.495s, table=0, n_packets=0, n_bytes=0, priority=100,ip,nw_src=10.123.123.1 actions=set_field:10.10.10.1->ip_src,goto_
table:1
 cookie=0x0, duration=130.495s, table=0, n_packets=0, n_bytes=0, priority=100,ip,nw_dst=10.10.10.1 actions=set_field:10.123.123.1->ip_dst,goto_
table:1
 cookie=0x0, duration=130.495s, table=0, n_packets=0, n_bytes=0, priority=0 actions=goto_table:1
 cookie=0x0, duration=130.495s, table=1, n_packets=0, n_bytes=0, priority=0 actions=goto_table:2
 cookie=0x0, duration=130.495s, table=2, n_packets=0, n_bytes=0, priority=100,tcp,nw_dst=10.123.123.1 actions=set_queue:7,output:5
 cookie=0x0, duration=130.494s, table=2, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER:65535
```

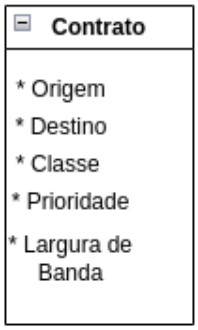
Fonte: (Próprio Autor, 2022).

4.2.3 Cliente/Servidor de Contratos

O FLOWPRI-SDN implementa uma forma de emissores receberem os recursos de rede necessários para seu fluxo de dados trafegar com QoS baseado em um mecanismo de contrato. O contrato contém dados em termos de largura de banda, classe de serviço, prioridade e endereços IP origem e destino do fluxo que será emitido (ver Figura 21). Com isso, fica acordado que o fluxo emitido necessita exatamente dos recursos definidos no contrato para trafegar com qualidade, enquanto que o *framework* se compromete em alocar a largura de banda necessária, sempre que

for possível. Desta forma, é assumido que os emissores de fluxos de dados tenham capacidade de identificar seus requisitos de emissão de tráfego, seja por eles mesmos ou por uso de dispositivos intermediários como em (COSTA; OLIVEIRA, 2020).

Figura 21 – Contrato.



Fonte: (Próprio Autor, 2022).

O contrato é um mecanismo usado exclusivamente para identificação dos fluxos no domínio. A alocação de banda para os fluxos depende do módulo de priorização e de gerenciamento de largura de banda (G-BAM). Desta forma, mesmo fluxos que estabeleceram um contrato podem ser negados de trafegar com QoS em caso de ter baixa prioridade ou falta de recursos na rede.

Os *hosts* que não precisam de recursos garantidos ou que não conheçam seus requisitos, podem acessar a rede sem necessidade de se comunicar com o *framework* para estabelecer contrato de QoS. Neste caso, o tráfego será automaticamente classificado como fluxo de dados da classe *best-effort* e será encaminhado pela fila 7 de alguma interface, sem reserva de recursos. A ideia principal é permitir que qualquer aplicação possua seus requisitos atendidos conforme sua ordem de prioridade, mas também seja compatível com serviços baseados em *best-effort*. Assim, no contexto de cidade inteligente se assume que apenas fluxos de aplicações de cidades inteligentes possam ser categorizadas como tráfego de nível médio e alto, no caso de fluxos das classes de tempo-real e não-tempo-real.

Para receber os contratos de QoS, o FLOWPRI-SDN possui dois *sockets* um ouvindo conexões TCP, um na porta 4444 para *hosts* do domínio, e outro na porta 8888 para controladores de outros domínios. O Algoritmo 2, descreve o funcionamento.

Algorithm 2 SERVIDOR DE CONTRATOS - RECEBIMENTO

```
Entrada:
início
    repita
        Esperar uma conexão
        Receber o contrato
        Obter os switches da rota entre os endereços do contrato
        se Já existe um contrato idêntico então
            se A conexão é de um host (porta 4444) então
                Criar um pacote ICMP 15 anunciando o contrato
                Injetar no último switch da rota
                Fim da função
            fim
            Fim da Função
        fim
        se Já existe um contrato com os mesmos endereços, mas com requisitos diferentes então
            para Cada switch da rota faça
                Obter a porta destino correspondente ao destino do contrato
                Remover as regras correspondentes da instância do switch
                Remover as regras correspondentes do switch OVS
                Executar a rotina G-BAM e salvar as ações
            fim
        fim
        se Todos os switches aceitaram o fluxo então
            Criar a regra meter para o fluxo
            Executar as ações de criação e remoção de regras
            No primeiro switch da rota é criada a regra de marcação do DSCP correspondente
        fim
    até Verdadeiro;
fim
```

Primeiro uma conexão é aguardada, então, o contrato é recebido. Com contrato, é verificado se já não existe um contrato correspondente, se existe, então é uma atualização de contrato. Nesse caso, são obtidas as instâncias de *switches* que fazem parte da rota entre os endereços do contrato no domínio e em cada um deles são removidas as regras antigas e criadas as novas regras referentes a esse contrato, tanto nas instâncias quanto nos *switches* OVS. Caso esse contrato de QoS seja completamente novo, então não é necessário remover regras antigas.

A partir disso, cada *switch* passa pela rotina G-BAM para alocar o fluxo seguindo o novo contrato (explicado na subseção 4.2.6). Se todas as instâncias de *switches* aceitarem o fluxo, são criadas as regras de classificação e encaminhamento nas tabelas OVS dos *switches*, para marcar os pacotes com o código *DSCP* correspondente de modo a filtrar os pacotes para encaminhar na fila correta. Isso finda a rotina para recebimento de contratos de controladores, a diferença é que na versão para receber contratos de *QoS* de *hosts*, após alocar o fluxo é gerado o anúncio do contrato por meio de um pacote ICMP 15 desse contrato (explicado na subseção 4.2.5). A parte

de envio de contrato é bem mais simples, pois apenas ocorre a transferência do contrato.

4.2.4 Tratamento de *Packet_in*

Pacotes que não possuem *matching* com as regras das tabelas de encaminhamento dos *switches* são enviadas ao FLOWPRI-SDN gerando o evento *packet_in*. Com esse evento, o *framework* é capaz de identificar fluxos que já possuem um contrato de QoS e devem receber os recursos solicitados, além de realizar o envio do contrato para outros domínios.

No tratamento desse evento, são obtidas as informações do cabeçalho do pacote e do *switch* OVS que gerou o evento. Pacotes especiais do tipo ICMP 15 e 16, são tratados pela parte responsável pelo protocolo de anúncio/solicitação de contratos (explicado na subseção 4.2.5), enquanto os demais tipos de pacotes são testados nos contratos conhecidos, conforme mostra o Algoritmo 3.

Neste último caso, os endereços IPv4 origem e destino dos pacotes são procurados nos contratos armazenados no *framework*, de forma a se encontrar uma correspondência. Em caso positivo, são obtidas as instâncias de *switches* que fazem parte da rota entre os *hosts* do contrato a partir do *switch* que gerou o *packet_in*. Cada instância *switch* da rota obtida é testada no módulo G-BAM e se todos aceitam o fluxo, ou seja, se possuem recursos para garantir os requisitos do fluxo, as regras necessárias são criadas.

Primeiro, em cada *switch* OVS são criadas as regras *meter* para limitar a largura de banda dos fluxos no valor definido pelo contrato de QoS. As regras *meter* são por fluxo, então cada uma tem um identificador único, baseado na combinação dos endereços IP de origem e destino do contrato. Assim, cada regra gerada pelo G-BAM é executada, criando e removendo fluxos tanto na lista de regras da fila correspondente na instância *switch* do FLOWPRI-SDN, quanto na tabela de fluxos do *switch* OVS. As regras criadas nas tabelas de fluxos dos *switches* OVS são associadas as regras *meter* criadas anteriormente e utilizam do código *DSCP* do contrato de QoS juntamente com os endereços IP de origem e destino, para identificar o fluxo e encaminhar a uma fila específica da porta de saída destino. Por fim, no primeiro *switch* da rota é criada a regra de classificação para marcar os pacotes desse fluxo com o código DSCP.

No caso do pacote não ter correspondência nos contratos, então esse fluxo é *best-effort* ou de controle. Nesse caso, são obtidas as instâncias de *switches* da rota e são criadas as regras de marcação (no primeiro) e encaminhamento (todos), apenas nas tabelas do *switch* OVS com o DSCP da classe de fluxos *best-effort* sem limitar a largura de banda.

Algorithm 3 TRATADOR DO EVENTO *packet_in*

Entrada: *datapath*,*pacote*

início
- Obter o identificador do *switch* que gerou o evento
- Obter os dados dos cabeçalhos do pacote
- Obter os endereços IP do cabeçalho IPv4
- **se** *Protocolo do pacote é ICMP código 15* **então**
  - Executar rotina de tratamento ICMP 15
  - Finalizar função
- **fim**
- **se** *Protocolo do pacote é ICMP código 16* **então**
  - Executar rotina de tratamento ICMP 16
  - Finalizar função
- **fim**
- Consultar todos os contratos de QoS e verificar se o endereço IP origem e destino tem correspondência
- **se** *Existe um contrato com correspondência* **então**
  - Obter as instâncias de *switches* da rota entre IP origem e destino, partindo do *switch* que gerou o evento
  - **para** *Cada switch da rota* **faça**
    - Obter a porta destino para o pacote
    - Executar o algoritmo G-BAM e salvar as ações de criação e remoção de regras do *switch*
  - **fim**
  - **se** *Todos os switches da rota aceitaram o fluxo* **então**
    - Criar as regras meter em cada *switch*
    - **para** *Cada ação salva* **faça**
      - Executar a ação criando ou removendo regras da instância do *switch* no FLOWPRI-SDN e no *switch* OVS
    - **fim**
    - Criar a regra de classificação no primeiro *switch* da rota (marcar com DSCP correspondente)
    - Criar um pacote ICMP 15 com origem o IP do controlador e destino o IP destino do pacote
    - Adicionar o IP origem do pacote no campo de dados do ICMP
    - Injetar o ICMP 15 no último *switch* da rota
  - **fim**
  - Fim da função
- **fim**
- //se chegou aqui então é um fluxo de *best-effort*
- Obter os *switches* da rota entre IP origem e destino, partindo do *switch* que gerou o evento
- Primeiro *switch*: criar regra de marcação para IP origem, IP destino, DSCP *best-effort*
- **para** *Cada switch da rota* **faça**
  - Obter a porta de saída para o IP destino
  - Criar regra de encaminhamento conforme os endereços IP, código DSCP de *best-effort*, fila e porta de saída
- **fim**

fim

4.2.5 Protocolo de Anúncio/Solicitação de Contratos

O protocolo de anúncio/solicitação de contratos é o mecanismo de disseminação de informações de QoS de fluxo para domínios compatíveis com o FLOWPRI-SDN, de modo que os domínios que fazem parte da rota entre os endereços IPv4 origem e destino descritos no contrato, possam implementar as mesmas políticas de QoS. Esse protocolo envolve quatro etapas e é ativado sempre que um contrato de QoS é criado ou um fluxo que já estabeleceu um contrato é identificado em um evento de *packet_in*, sendo eles: o anúncio do contrato de QoS estabelecido utilizando um pacote ICMP 15, a solicitação de contratos de QoS utilizando um pacote ICMP 16, o envio do contrato de QoS solicitado e o seu recebimento no *framework* solicitante.

Uma forma de ativação envolve o estabelecimento do contrato de QoS descrito na subseção 4.2.3. Após salvar o contrato e criar as regras necessárias, ocorre o anúncio do contrato de QoS ao longo dos domínios da rota entre os endereços do contrato, por meio do envio de um pacote ICMP 15. A outra forma foi descrita na subseção 4.2.4, quando o pacote que gerou o evento possui correspondência com algum contrato, mas as regras para ele se expiraram, então são recriadas e o contrato é anunciado.

O pacote ICMP 15 é enviado com o endereço IPv4 do *framework* que está anunciando no campo de endereço de origem do pacote, enquanto que o endereço de destino do pacote, é marcado com o endereço IPv4 de destino do contrato anunciado. O endereço IPv4 de origem do contrato é colocado no campo de dados do pacote ICMP 15. Desta forma, os *frameworks* que analisarem o pacote, conseguem obter os endereços do contrato de QoS anunciado e o endereço IPv4 do *framework* que realizou o anúncio, para realizar a solicitação.

Deste modo, quando um pacote ICMP 15 entra no domínio gerenciado por um *framework* FLOWPRI-SDN, esse pacote é analisado pelo primeiro *switch* OVS da borda e vai gerar um evento *packet_in* enviando o pacote para ser analisado no FLOWPRI-SDN.

No tratamento desse evento, o FLOWPRI-SDN do domínio obtém o endereço IPv4 do *framework* emissor do ICMP 15 e os endereços do contrato de QoS anunciados, que estão nos campos do cabeçalho ICMP 15. São verificados se os endereços correspondem a algum contrato de QoS que está armazenado, caso tenha, é obtido o código *DSCP* desse contrato. Deste modo, o *framework* cria um pacote ICMP 16, de resposta, onde o endereço origem do pacote é marcado com o IP do FLOWPRI-SDN do domínio e o endereço destino do pacote é marcado com o mesmo endereço origem do pacote ICMP 15 recebido. Neste pacote ICMP 16, no campo de dados é colocado os endereços IPv4 de origem e destino do contrato de QoS anunciados, junto com o código DSCP encontrado antes, caso existir.

A partir disso, são obtidos as instâncias de *switches* da rota entre o *framework* do domínio e o emissor do ICMP 15. Nesses *switches* são criadas as regras de ida e volta para o recebimento do contrato de QoS solicitado nas tabelas de fluxo OVS. Nos *switches* OVS, primeiro e último da rota, são criadas as regras de classificação, para marcar os pacotes com o DSCP da classe de tráfego de controle, juntamente com as regras de encaminhamento necessárias. Desta forma, o

pacote ICMP 16 criado anteriormente é injetado no *switch* OVS que gerou o evento *packet_in* e o pacote é enviado para o FLOWPRI-SDN que anunciou o contrato de QoS. Por último, são obtidas as intâncias de *switches* que fazem parte da rota dos endereços origem e destino do pacote ICMP 15 recebido, para reinjeta-lo no último *switch* OVS, de modo a dar sequencia ao protocolo.

Da mesma forma, quando um pacote ICMP 16 é recebido em um *switch* de um domínio gerenciado por um *framework* FLOWPRI-SDN, esse pacote é identificado como sendo uma solicitação de contrato de QoS e o pacote é enviado para o controlador gerando um evento *packet_in*. No evento, o tratamento do pacote ICMP 16 tem dois comportamentos: quando o pacote ICMP 16 chega no FLOWPRI-SDN destino e quando chega em um FLOWPRI-SDN que não é o destino.

Quando o ICMP 16 chega no FLOWPRI-SDN destino, são verificados os valores do campo de dados do pacote ICMP 16 para obter os endereços referentes ao contrato desejado. Assim, se o DSCP obtido do pacote for igual ao contrato anunciado, não se envia o contrato. Em caso contrário, são obtidas as instâncias de *switches* da rota entre os endereços origem e destino do pacote ICMP 16, para criar as regras de classificação e encaminhamento do tipo controle em cada um e poder enviar os contratos. Após isso, o contrato é enviado para o endereço IP de origem do ICMP 16, para uma porta TCP reservada no FLOWPRI-SDN (8888), que escuta apenas controladores.

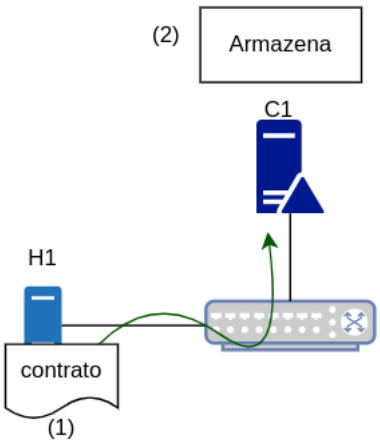
No caso de o *framework* não ser o destino do ICMP 16, então este é um controlador de um domínio que faz parte da rota na direção do destino e consequentemente os pacotes referentes ao contrato de QoS anunciados devem passar pelos *switches* desse domínio para chegar em seu destino. Desta forma, é necessário criar as regras para que esses fluxos de ida e volta sejam tratados como de controle, pois fazem parte do protocolo de anúncio e requisição de contratos.

Deste modo, são obtidas as instâncias de *switches* da rota entre os endereços origem e destino do pacote ICMP 16 para criação das regras de classificação e encaminhamento, com DSCP de controle, de modo que o tráfego seja encaminhado pela fila de controle em cada *switch* OVS da rota. Por fim, o pacote ICMP 16 é reinjetado no *switch* OVS mais próximo da borda de destino.

A Figura 22 exemplifica o que ocorre no início do processo de anúncio de contratos, excluindo os processos de criação de regras. Ele se inicia no momento em que um contrato é criado pelo *host* $H1$ no domínio do controlador $C1$, que informa os requisitos de tráfego para o envio de dados em direção ao *host* $H2$, de outro domínio. Assim, o controlador $C1$ (FLOWPRI-SDN) armazena tal contrato, caso haja aceitação das condições de QoS nesse domínio. A partir da aceitação do fluxo nos *switches* de $C1$, é enviado um pacote *ICMP* 15 com origem $C1$ e destino $H2$ para anunciar o contrato de QoS estabelecido para outros domínios compatíveis, conforme mostra a Figura 23. Então, no domínio de $C2$, é verificado o pacote de anúncio de contratos *ICMP* 15 e é criado um pacote de resposta *ICMP* 16, solicitando o contrato, com origem o endereço de $C2$ e destino o endereço de $C1$. Além disso, o pacote *ICMP* 15 é reinjetado

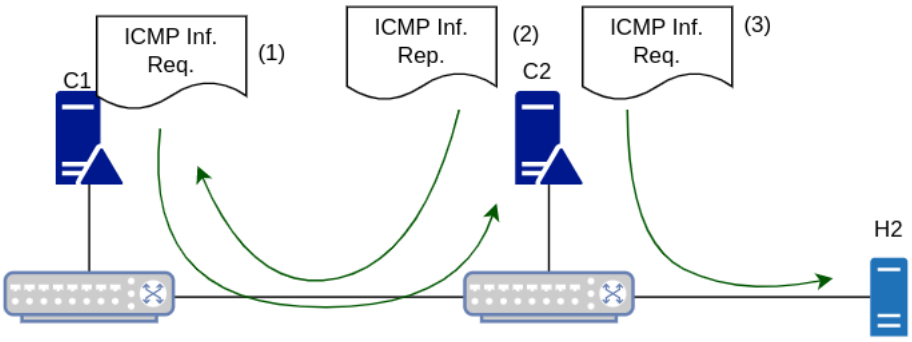
para dar sequência ao protocolo.

Figura 22 – Estabelecimento de contrato.



Fonte: Próprio autor, 2022.

Figura 23 – Troca de mensagens ICMP.



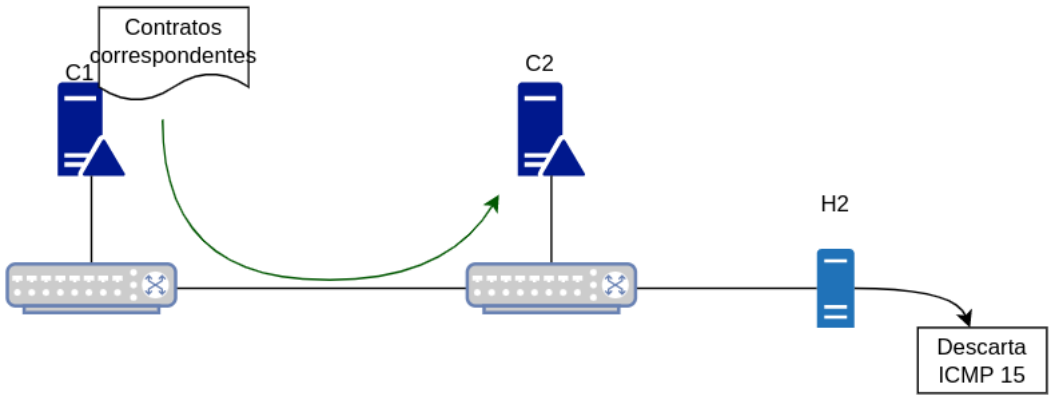
Fonte: Próprio autor, 2022.

Quando $C1$ recebe o pacote *ICMP* 16 de resposta, ele envia o contrato desejado em direção a $C2$, como é mostrado na Figura 24. Assim $C2$ armazena o contrato e cria as regras necessárias nos *switches* de seu domínio. No entanto, o pacote *ICMP* 15 em algum momento alcança o *host* $H2$. Deste modo, $H2$ pode identificar que algum *host* pretende enviar dados com QoS para seu endereço e preparar um contrato para o tráfego de retorno. Caso contrário, por serem pacotes de um mecanismo de controle descontinuado, por padrão $H2$ descarta o pacote.

4.2.6 G-BAM

O modelo de gerenciamento de largura de banda implementado pelo FLOWPRI-SDN nos *switches* OVS do seu domínio para aceitação de fluxos e alocação de recursos segue a estratégia definida pelo modelo de alocação de largura de banda G-BAM. O modelo G-BAM defende implementar estratégias de alocação de recursos que envolvam três tipos de compartilhamento de largura de banda entre classes de serviços: uma parte da largura de banda é privada e não é compartilhada com outras classes, parte da largura de banda é reservada para classes de serviço

Figura 24 – Envio do contrato.



Fonte: Próprio autor, 2022.

de maior prioridade e a largura de banda excedente deve ser compartilhada com classes de menor prioridade, por fim, parte da largura de banda é reservada para classes de serviços de menor prioridade e devem compartilhar largura de banda excedente com classes de maior prioridade.

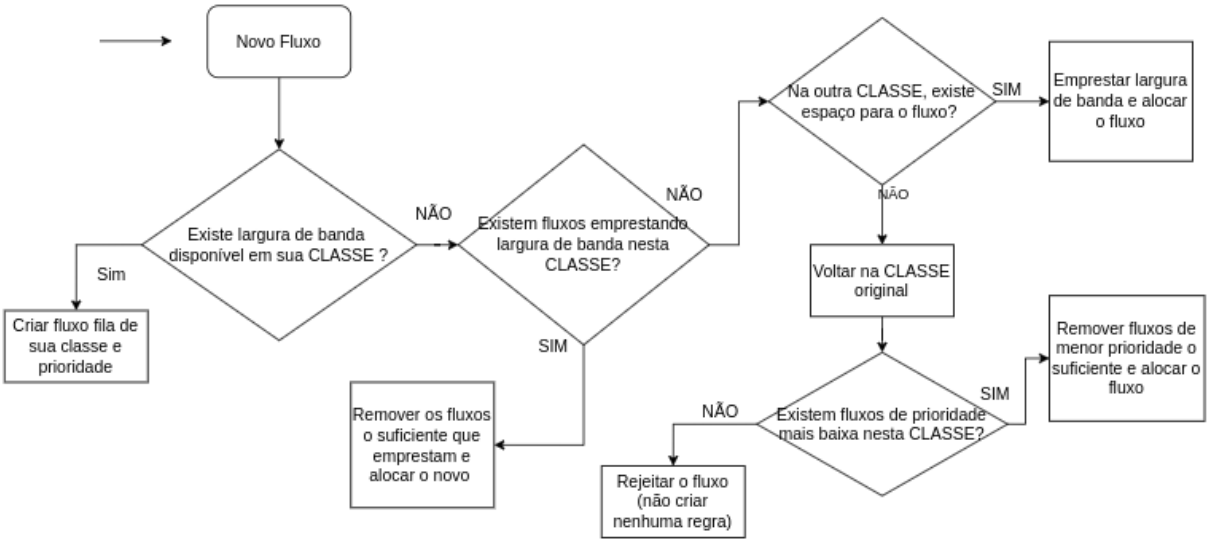
Utilizando os recursos das filas *HTB*, o FLOWPRI-SDN implementa quatro classes de serviços. As classes de fluxos de tempo-real e de não-tempo-real compartilham largura de banda entre fluxos de baixa, média e alta prioridade, ou seja, as filas dessas classes podem utilizar a largura de banda que estiver disponível. A classe de controle não empresta largura de banda de outras classes e a classe de serviço *best-effort* permite que sua fila utilize toda a largura de banda não utilizada. Deste modo, essa estratégia de divisão de largura de banda se encaixa na ideia geral proposta pelo modelo *G-BAM*.

O FLOWPRI-SDN alia a estratégia de divisão de largura de banda com uma estratégia de alocação reservada de largura de banda. Essa estratégia é aplicada para alocação de fluxos com reserva de recursos para as classes de serviços de fluxos de tempo-real e de fluxos de não-tempo-real. Neste caso, o uso da largura de banda emprestada entre as filas dessas classes é controlada pelo FLOWPRI-SDN e não pelo HTB, o HTB somente garante a disponibilidade mínima da largura de banda.

No FLOWPRI-SDN, o uso combinado dos dois modelos é chamado de *G-BAM*. A rotina G-BAM busca posicionar o fluxo de dados testado, na fila correspondente à prioridade e classe de serviço definidas no contrato de QoS. No entanto, caso a classe de fluxos de tempo-real não tenha largura de banda disponível o suficiente para alocar o novo fluxo, esse fluxo de dados pode emprestar largura de banda de forma reservada da outra classe de serviço mantendo a mesma prioridade definida e vice-versa, conforme o algoritmo descrito na Figura 25.

A entrada do algoritmo é uma instância de *switch* e um fluxo de dados, com sua largura de banda, prioridade e classe descritos pelo contrato de QoS admitido. Primeiro, se verifica se na porta destino do *switch*, para onde o fluxo será encaminhado, a classe de serviço do fluxo possui largura de banda suficiente para esse fluxo. Caso possua, é criada a ação de criação de regra de encaminhamento com *matching* os endereços *IP* definidos no contrato e o código *DSCP*, para

Figura 25 – G-BAM.



Fonte: (Próprio Autor, 2022).

que saia pela porta correta e utilize a fila correspondente a sua classe e prioridade. O desconto e incremento da largura de banda disponível das classes de serviços ocorrem sempre quando uma regra é criada ou removida de uma de suas listas de regras na instância do *switch*. Deste modo, a função G-BAM retorna a lista de ações que devem ser executadas na instância de *switch* testada.

Caso um fluxo precise emprestar largura de banda de outra classe, o fluxo é salvo na ação, com a *flag* "emprestando"ligada, que instrui na execução da regra armazenar na fila de prioridade da classe de serviço contrária (tempo-real ou não-tempo-real), descontando de sua largura de banda disponível, mas mantendo o mesmo cógido DSCP original para o contrato de QoS do fluxo.

Ao final da função, a saída é um vetor de instruções que contém quais regras devem ser deletadas e quais devem ser criadas nas filas da instância de *switch* testada no algoritmo e na tabela de fluxo OVS desse *switch*. Deste modo, na versão desenvolvida do modelo de alocação, fluxos não podem ser alocados metade em uma classe e metade em outra, ou seja, a classe onde se posiciona o fluxo deve conter espaço suficiente para alocar inteiramente o fluxo.

4.2.7 Sincronização de Regras

Como o FLOWPRI-SDN armazena os estados dos *switches* e os utiliza para realizar a alocação de recursos, é implementado um mecanismo de sincronização de regras entre FLOWPRI-SDN e *switches* OVS. Sempre que uma regra de encaminhamento é criada ela é definida com um *idle_timeout*, no caso de regras para as classes tempo-real e não-tempo-real, ou com um *hard_timeout* nos outros casos. O *idle_timeout* determina que se uma regra ficar inativa por um tempo determinado essa regra deve ser removida do *switch* OVS, enquanto que o *hard_timeout* remove após o tempo determinado, independentemente de a regra estar sendo utilizada.

Para manter as regras sincronizadas na instância presente no FLOWPRI-SDN referente

ao *switch* OVS, a *flag send_flow_removed* é ativada pelo FLOWPRI-SDN quando se cria regras de encaminhamento nas tabelas dos *switches* OVS. Essa *flag* instrui os *switches* OVS para que, sempre que uma regra expirar, gerar o evento *flow_removed* em direção ao FLOWPRI-SDN, com as informações da regra removida. Neste caso, o FLOWPRI-SDN obtém a regra e identifica os endereços IP origem, destino e código *DSCP* do campo de *matching* da regra. Assim, se obtém as instâncias de *switch* do domínio, que fazem parte da rota entre os endereços IP da regra. Para cada instância, é encontrada a porta de saída para o endereço destino da regra e nessa porta, são buscados nas listas de regras uma correspondência com a regra do evento. Quando a correspondência for encontrada, a regra de encaminhamento é removida da instância do *switch*, juntamente com a regra *meter* envolvida, somando a quantidade de largura de banda que era utilizada pelo fluxo na largura de banda disponível para a classe de serviço.

4.3 CONSIDERAÇÕES DO CAPÍTULO

Neste Capítulo, foi definido o *framework* para tratar do gerenciamento e fornecimento de largura de banda e priorização para fluxos de dados que pode ser aplicado às cidades inteligentes. O FLOWPRI-SDN assume uma divisão da largura de banda em classes de serviços: Tempo-real, tráfego de fluxos não-tempo-real, fluxos não identificados (*best-effort*) e tráfego de controle de rede.

Com essas classes, foi proposto utilizar os conhecimentos dos documentos que estudaram as características dos fluxos de dados que trafegam na rede e seus requisitos de QoS para desenvolver uma modelagem de identificação de fluxos baseado no campo de cabeçalho DSCP dos pacotes IP. Esse campo permite definir códigos que representam os tipos de tráfego suportados pelo FLOWPRI-SDN, da mesma maneira que cada código representa a informação da classe de serviço do pacote, quantidade de largura de banda e priorização necessária.

Com isso, o FLOWPRI-SDN engloba os mecanismos de admissão de fluxos de dados e estabelecimento de acordo de requisitos, módulo de gerenciamento de largura de banda, baseado na estratégia *G-BAM*, e divisão da largura de banda em classes de serviços, por meio da criação de filas HTB nas portas de saída dos dispositivos de rede. Além disso, com as filas HTB foi possível modelar uma estratégia de priorização, que é muito importante para os fluxos de dados de aplicações para cidades inteligentes.

Por fim, além de o FLOWPRI-SDN promover o gerenciamento de recursos e o tratamento de fluxos de dados em seu domínio, também foi proposto um mecanismo de colaboração para que as políticas empregadas sejam válidas ao longo do caminho. Para isso, o mecanismo envolve utilizar pacotes do protocolo ICMP 15 e 16, de modo realizar o anúncio e requisição de contratos de QoS, permitindo que controladores compatíveis ao longo do caminho entre os endereços IP de origem e destino do contrato possam aplicar as mesmas políticas de QoS. Deste modo, a abordagem proposta se mostra promissora e pode ser capaz de fornecer QoS próxima de fim-a-fim em cenários mais otimistas.

5 EXPERIMENTOS E RESULTADOS

Esta seção apresenta os testes realizados e resultados obtidos na avaliação do FLOWPRI-SDN. A máquina virtual utilizada nos testes possui 4 CPUs e 2*Gb* de RAM. O sistema operacional utilizado foi o Ubuntu 20.04.1 LTS, *Kernel* $5.4.0-42-generic$.

Para realização dos testes, os cenários foram montados por meio de topologias de rede utilizado o simulador de redes Mininet. Este *software* é um emulador para implementar redes de diversos tamanhos, sobre os recursos limitados de um computador ou máquina virtual (KAUR; SINGH; GHUMMAN, 2014). Com isso, é possível emular os dispositivos de redes e topologias conforme um cenário real de um paradigma tradicional ou SDN. O Mininet permite criar e interconectar *switches* OVS, controladores OpenFlow, incluindo o Ryu e *hosts* de forma fácil e prática. A versão utilizada do Mininet foi a 2.3.0*b*2, enquanto a versão do Ryu foi 4.34 e do OVS foi 2.13.1.

O uso de simuladores é necessário na área de redes devido aos componentes e equipamentos serem caros e, muitas vezes, indisponíveis. Nesse sentido, o Mininet fornece uma API em Python e permite a configuração de dispositivos físicos de maneira virtual, implementando toda a rede na memória (OLIVEIRA et al., 2014). Os enlaces entre os *switches* podem ser configurados com diversas características inerentes aos contextos reais, como largura de banda, atraso e taxa de perda de pacotes.

Por meio do Mininet, os testes do FLOWPRI-SDN se concentraram em dois tópicos: teste de *overhead* inserido pelos procedimentos do *framework* e teste de caso de uso em um cenário de cidade inteligente. O teste de overhead busca mostrar quanto tempo é consumido para estabelecimento de um contrato de QoS e o tempo de envio de pacotes quando o contrato já está estabelecido. Por sua vez, o teste de caso de uso busca mostrar o que ocorre em um cenário com o *framework* vs sem o *framework*, onde um tráfego prioritário precisa trafegar com QoS e a rede está congestionada de tráfego *background* (*best-effort*). Portanto, as métricas observadas nos testes são: tempo de processamento, tempo de encaminhamento, largura de banda utilizada e QoS obtido.

Sendo assim, na seção 5.1, são apresentados alguns pontos importantes que devem ser levados em consideração quando aplicar o FLOWPRI-SDN sobre um ambiente simulado. Os testes de *overhead* causados pelo FLOWPRI-SDN são apresentados na seção 5.2. Na sequência, a seção 5.3 apresenta a aplicação do *framework* em um cenário de cidade inteligente.

5.1 AMBIENTE MININET

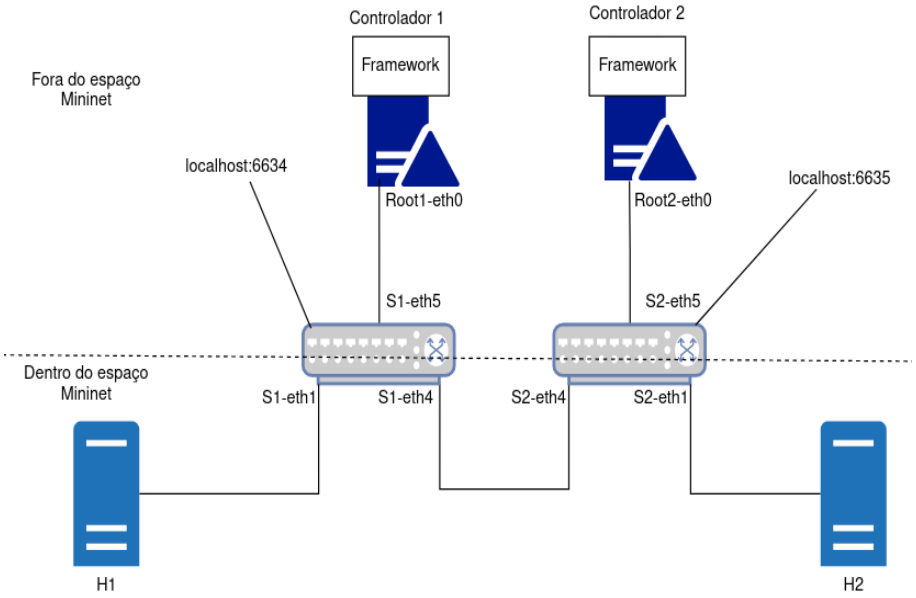
Para executar diversos controladores Ryu em uma mesma máquina virtual de modo que tenham acesso aos *hosts* da rede, é necessário usar artifícios para que cada controlador escolha a interface correta para comunicação. O ambiente Mininet consiste de uma aplicação isolada do espaço do usuário, onde são criados os *hosts*. A partir disso, podem ser criadas *bridges* OVS (*switches*) interconectadas, que são visíveis tanto no ambiente Mininet quanto fora dele.

No entanto, o controlador Ryu é uma aplicação que executa fora do ambiente Mininet, sendo acessível para os *switches* mas não pelos *hosts*.

Desta forma, não é possível executar um controlador Ryu dentro de um *host* do ambiente Mininet, pois este não conseguiria se conectar com a interface OpenFlow dos *switches* do domínio. Assim, para que seja possível se comunicar tanto com os *hosts* Mininet quanto com os *switches* OVSé necessário criar um *host* fora do ambiente Mininet.

Esse *host* possui uma interface que é visível fora do ambiente Mininet e que pode ser conectada a um *switch* OVS. Com isso, é possível executar o controlador Ryu (*framework*), de modo que possa programar os *switches* OVS usando OpenFlow e se conectar aos *hosts* Mininet por meio da interface do *host* criado. Isso permite que o *framework* possa escutar uma porta TCP e receber os contratos de QoS dos *hosts* Mininet. Além de permitir que *frameworks* se comuniquem por meio das interfaces de seus *hosts* por meio da rede Mininet e não pelo ambiente *linux*.

Figura 26 – Ambiente Mininet.



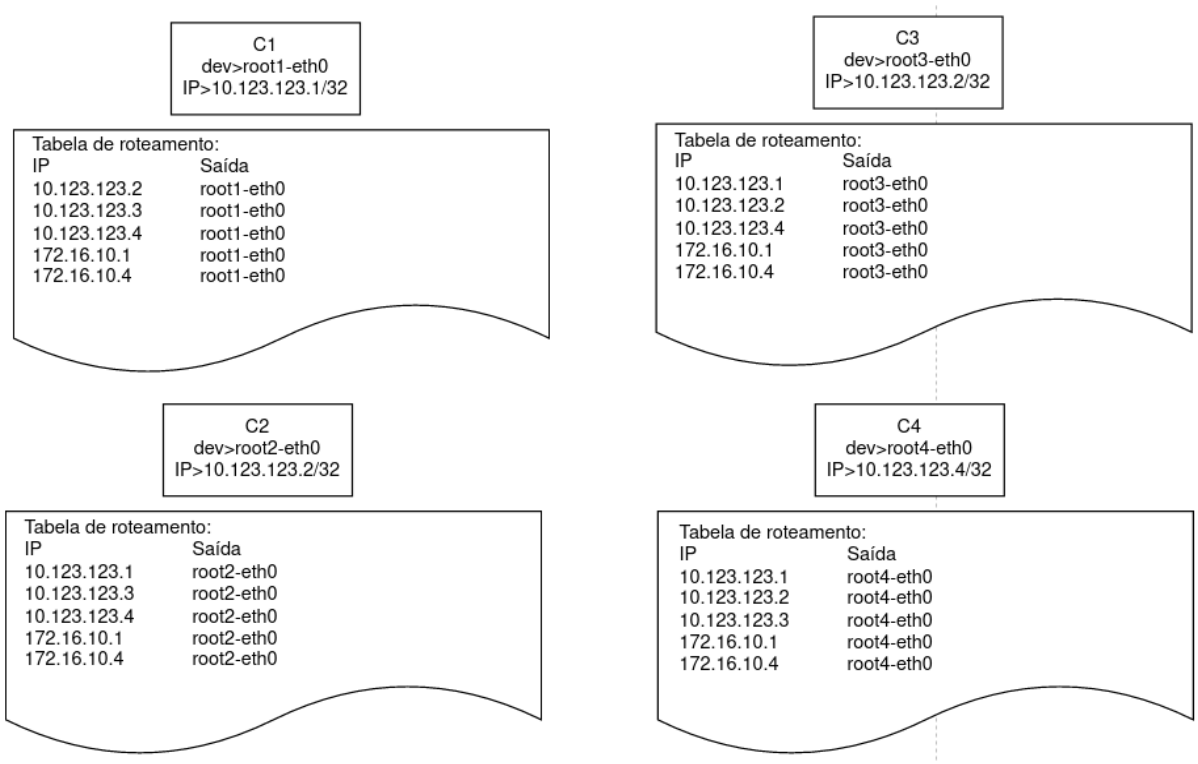
Fonte: (Próprio Autor, 2022).

No entanto, os *hosts* criados fora do ambiente Mininet utilizam todos a mesma tabela de roteamento e de *arp*. Com isso, a parte de comunicação entre *hosts* Mininet e o *host* controlador ficaria comprometida, uma vez que os pacotes vindo do ambiente Mininet não poderiam ser respondidos corretamente pois não seria possível escolher a interface correta para enviar pacotes de resposta.

Esses problemas foram resolvidos criando endereços IP fictícios para cada interface de *host* de controlador. A Figura 27 mostra como seria a tabela de roteamento dos controladores caso eles fossem, de fato, *hosts* isolados.

A Figura 28 apresenta a metodologia de traduções de endereços para que todos os controladores utilizem a mesma tabela de roteamento e estejam em uma mesma máquina. Neste

Figura 27 – Tabela de roteamento convencional.



Fonte: (Próprio Autor, 2022).

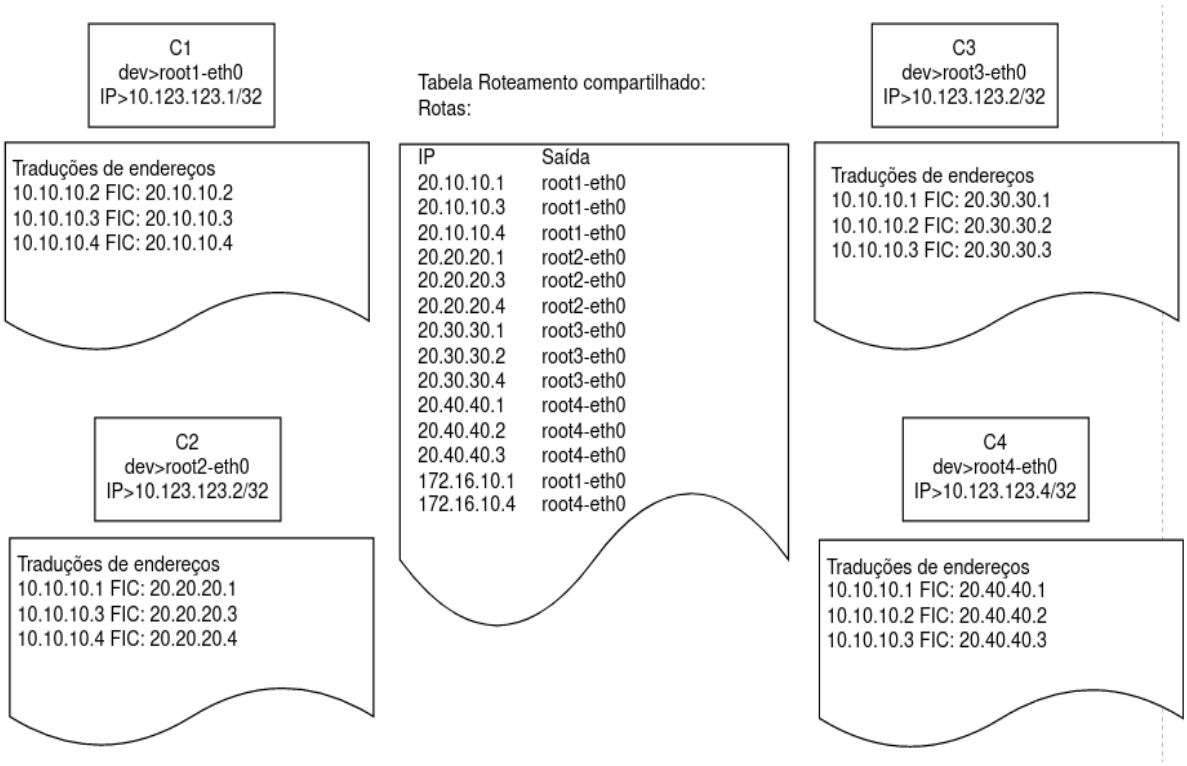
caso, cada *host* de controlador possui sua interface e seu endereço IP, que é visto como endereço local. No entanto, dentro do ambiente Mininet, os endereços IP dos *hosts* de controladores são traduzidos para o modelo 10.10.10.*X*, onde X é o identificador do controlador.

Para isso, foi implementada uma camada extra nas tabelas de fluxo dos *switches* que se conectam diretamente ao FLOWPRI-SDN por um salto. Nesses *switches*, a tabela de fluxos 0 é de pré-marcação, a tabela de fluxos 1 é a de classificação e a 2 é a de encaminhamento, diferente do que ocorre com os demais *switches*. A tabela de pré-marcação realiza as trocas dos endereços IP origem e destino necessárias, antes de testar nas outras regras da tabela de fluxo do *switch* OVS.

Os endereços definidos para as interfaces dos *hosts* de controladores, são do tipo 10.123.123.*X*. Para realizar as traduções necessárias, são instaladas regras específicas no *switch* mais próximo do *host* de cada controlador para receber pacotes corretamente. A tabela de pré-marcação remarca com endereço origem 10.10.10.*X* os pacotes que saem dos controladores e remarca o endereço destino de pacotes que chegam nos controladores com 10.123.123.*X*, onde *X* é o identificador do *host* controlador que recebe o pacote. Isso faz com que o *host* controlador receba os pacotes como se tivesse o endereço destino dentro do espaço de endereçamento Mininet e envie com o endereço reconhecido pelos dispositivos da topologia Mininet.

Além de traduções para receber pacotes corretamente no *host* de controlador, é necessária tradução de endereços para comunicar entre *hosts* de controladores. Neste caso, se todos os controladores recebessem pacotes marcados como 10.10.10.1, não seria possível identificar qual

Figura 28 – Tabela de roteamento com tradução de endereços.



Fonte: (Próprio Autor, 2022).

a interface usar para responder por causa da tabela de roteamento compartilhada. Então, para não cair no mesmo problema, os *hosts* de controladores também precisam se ver com endereços fictícios.

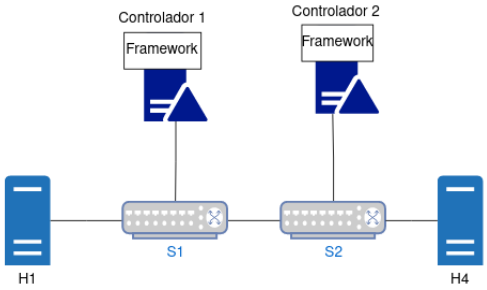
Para isso, outra regra é criada na tabela de pré-marcação. Essa regra remarca o endereço origem de pacotes que chegam no *switch* de um *host* controlador para outro, do formato 10.10.10.Y para 20.X0.X0.Y, onde X é o identificador do *host* controlador que recebe e Y é o identificador do *host* controlador que envia. Assim, internamente ao ambiente Mininet cada *host* controlador pode ter a sua entrada na tabela de roteamento para tratar pacotes de outros *hosts* de controladores e ser capaz de se comunicar sem perceber a diferença, habilitando testes de cenários multi-controladores.

5.2 TESTE DE OVERHEAD

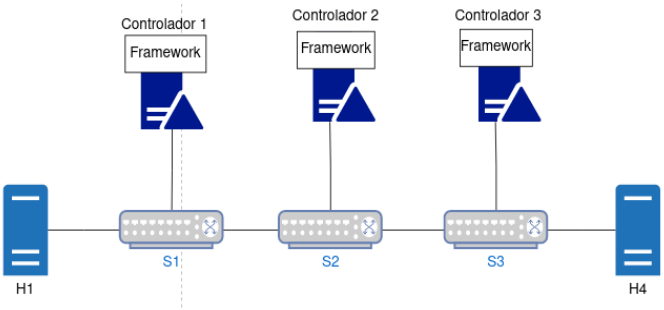
As topologias usadas para testar o *overhead* são apresentadas na Figura 29. Foram feitos testes com cenários multi-controladores, em que cada um um administra um *switch* em seu domínio de gerenciamento. Para a configuração de testes, em cada cenário, um *host* está conectado ao domínio de uma ponta, enquanto que do outro lado se conecta um *host* que representa a nuvem, para que se possa testar o *overhead* do FLOWPRI-SDN na comunicação entre eles.

Primeiro, é feito o teste de *overhead* para estabelecimento de contrato. Neste caso, o *host*

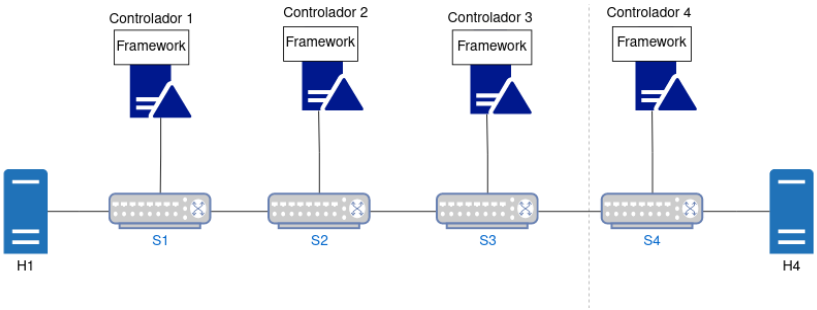
Figura 29 – Cenários de teste de *Overhead*



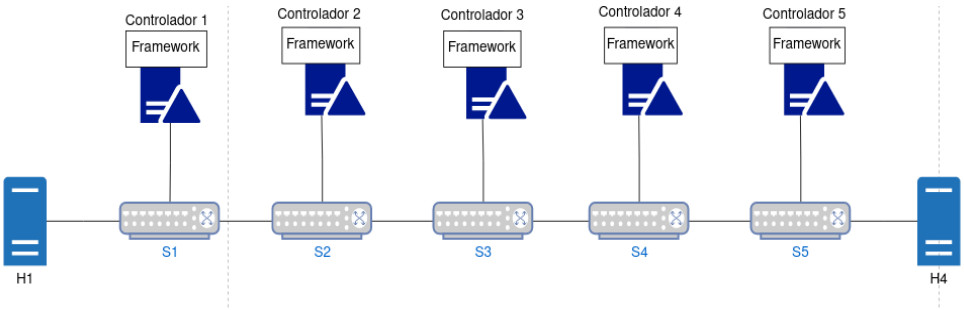
(a) Cenário com 2 domínios



(b) Cenário com 3 domínios



(c) Cenário com 4 domínios



(d) Cenário com 5 domínios

Fonte: (Próprio Autor, 2022).

$H1$ cria um contrato para envio de tráfego para a nuvem ($H4$) e o estabelece com o FLOWPRI-SDN mais próximo ($C1$), que vai realizar todos os procedimentos já descritos para anunciar e enviar o contrato para os controladores dos domínios intermediários e criar as regras necessárias. Os tempos dos procedimentos são medidos dentro de cada função usando a biblioteca *Python datetime*, apresentando o *timestamp* do início da função e do fim.

Desta forma, no *framework* os tempos consumidos para executar as seguintes funções foram medidos: função para receber e estabelecer contrato de QoS com *hosts* (*server_host*), função que trata o recebimento de um pacote ICMP 15, função que trata o recebimento de um pacote ICMP 16 e envia um contrato de QoS para outro *framework*, e a função que recebe um contrato de outro *framework* (*server_controller*). Os demais processos de alocação e remoção de regras de fluxos são internos a essas funções e estão sendo contabilizadas.

Assim, o tempo de processamento de todos os *frameworks* envolvidos na comunicação são medidos e somados para verificar o tempo médio consumido em cada processo e perceber a distribuição do processamento entre as funções. Nesse teste, é possível perceber o tempo total consumido com processamento nos *frameworks* durante um estabelecimento de contrato de QoS.

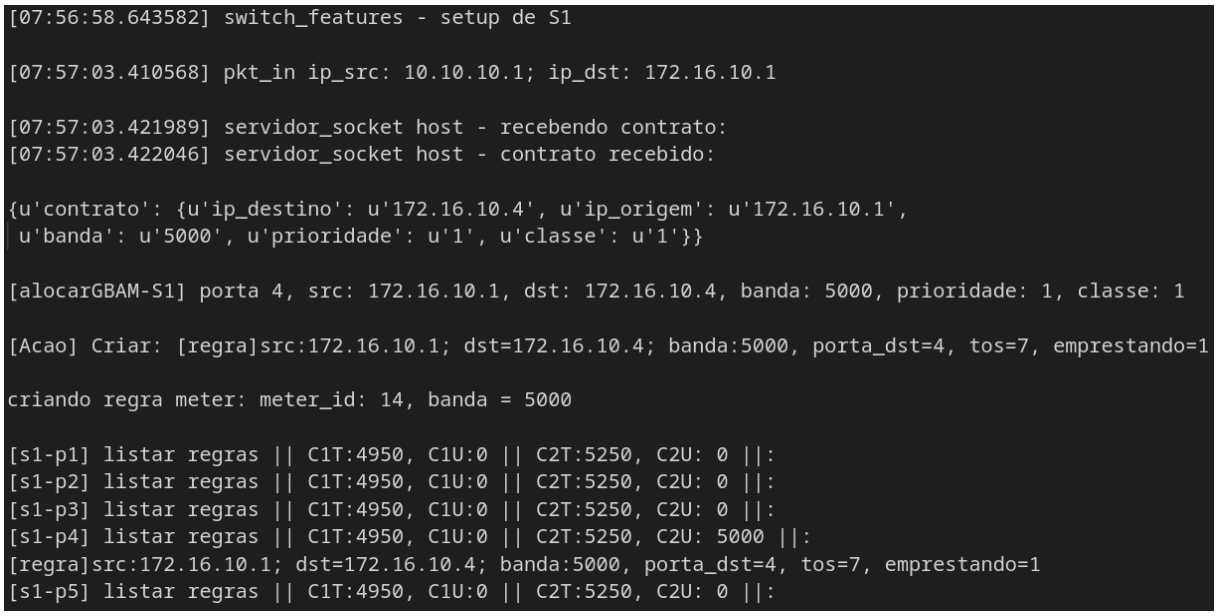
O tempo percebido do estabelecimento do contrato de QoS é o tempo real que o *host* deve esperar para que seu fluxo possa ser tratado conforme os requisitos solicitados. O percebido é medido a partir do primeiro pacote que sai da interface do *host* $H1$ em direção ao *framework* da borda $C1$, até o fim do procedimento de recebimento do contrato no último controlador da rota. Após isso, as regras em todos os domínios da rota devem ter sido criadas, caso possível, para que o fluxo de $H1$ trafegue com QoS até $H4$.

A Figura 30, mostra os *logs* de quando o *host* $H1$ estabelece o contrato com o FLOWPRI-SDN do controlador $C1$. Então, $C1$ cria as regras de marcação, encaminhamento e de *meter* nos *switches* da rota dentro de seu domínio, finalizando com o envio de um pacote ICMP 15 anunciando o contrato. A Figura 31, mostra os *logs* do momento que $C2$ recebe e trata o ICMP 15, responde com um ICMP 16 e reinjeta o ICMP 15 em outro domínio para dar sequencia no processo de anuncio de contrato. Na sequencia, $C2$ recebe o contrato e cria as regras necessárias nos *switches* da rota. A Figura 32 apresenta os *logs* do momento em que $C1$ recebe o ICMP 16 de $C2$ e envia o contrato solicitado.

Para cada topologia, foram realizadas 10 tomadas de testes, nos quais os tempos de processamento individuais de cada *framework* foi coletado e agrupado. A Figura 33 mostra o tempo consumido para cada etapa de processamento em cada topologia durante o estabelecimento do contrato de QoS. Como é possível observar, o tempo consumido pelo tratamento de pacotes ICMP 16 representa a maior porção de processamento. Isso se deve pelo fato de que o FLOWPRI-SDN que emitiu o primeiro ICMP 15 precisa responder o pacote ICMPs 16 de solicitação de contrato de cada FLOWPRI-SDN intermediário, criar as regras específicas para se comunicar com cada outro *framework* solicitante e enviar o contrato de QoS.

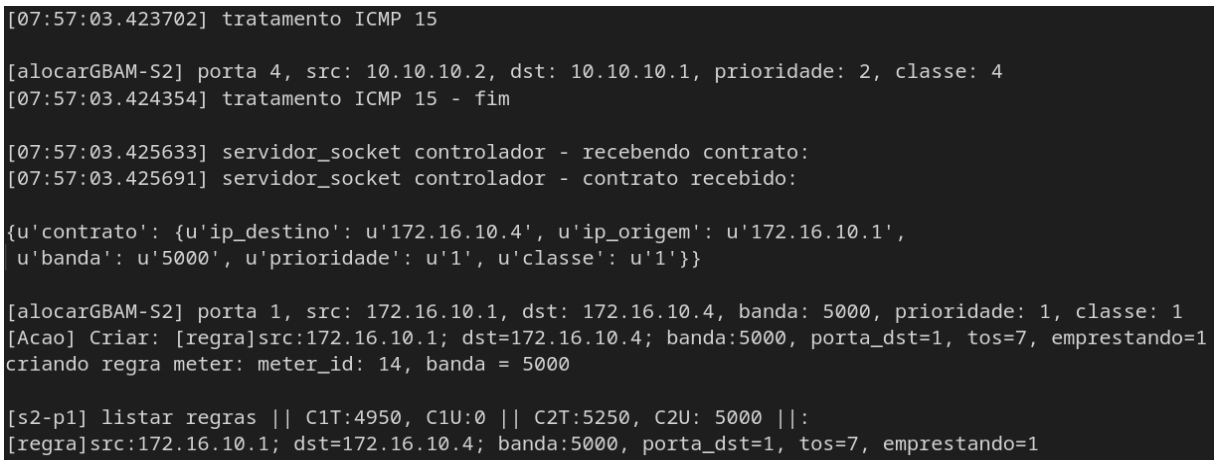
O tempo percebido, é o tempo entre o primeiro pacote que sai de $H1$ em direção a $C1$ para estabelecimento do contrato, até o final do procedimento para receber o contrato no último

Figura 30 – C1 configurando S1 e recebendo o contrato de H1.



Fonte: (Próprio Autor, 2022).

Figura 31 – C2 tratamento de ICMP 15 e recebimento de contrato de C1.



Fonte: (Próprio Autor, 2022).

framework da rota entre $H1$ e $H4$. Quando o último *framework* recebe o contrato de QoS, este cria as regras e aloca os recursos necessários para o fornecimento de QoS. Neste momento, todos os outros FLOWPRI-SDN intermediários já estão configurados e $H1$ pode enviar dados para $H4$ com os recursos reservados nos domínios gerenciados por *frameworks* FLOWPRI-SDN.

Os tempos percebidos foram obtido das 10 execuções anteriores. Esses valores são ligeiramente diferente do tempo de processamento em cada topologia, pois envolvem o tempo de processamento, juntamente com o tempo de comunicação entre *host* e *framework*, e o tempo de comunicação entre *frameworks*. A Figura 33 também mostra o tempo de processamento em cada topologia, para que o contrato estabelecido seja configurado em todos os *frameworks* da rota.

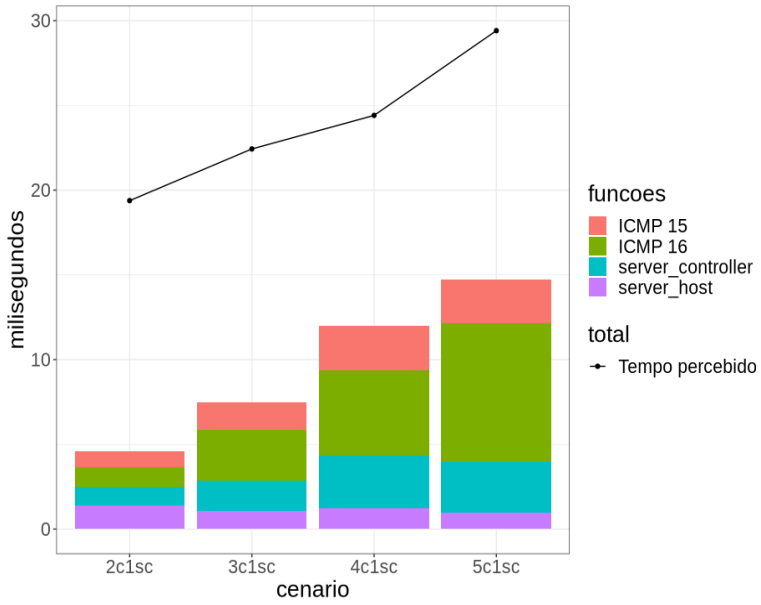
O tempo percebido obtido é sempre maior que o tempo de processamento, no entanto devido a questões de comunicação. Durante os testes, o tempo de processamento individual de

Figura 32 – C1 enviando contrato para C2.

```
[07:57:03.424837] pkt_in ip_src: 20.10.10.2; ip_dst: 10.123.123.1

[07:57:03.424878] tratamento ICMP 16 - controlador destino

[alocarGBAM-S1] porta 4, src: 10.10.10.1, dst: 10.10.10.2, prioridade: 2, classe: 4

[07:57:03.425449] tratamento ICMP 16 - controlador destino - fim

[07:57:03.425575] enviar contrato p/ 20.10.10.2

{u'contrato': {u'ip_destino': u'172.16.10.4', u'ip_origem': u'172.16.10.1',
 u'banda': u'5000', u'prioridade': u'1', u'classe': u'1'}}

[07:57:03.425647] enviar contrato p/ 20.10.10.2 - fim
```

Fonte: (Próprio Autor, 2022).

Figura 33 – Tempo de processamento e tempo real percebido.



Fonte: (Próprio Autor, 2022).

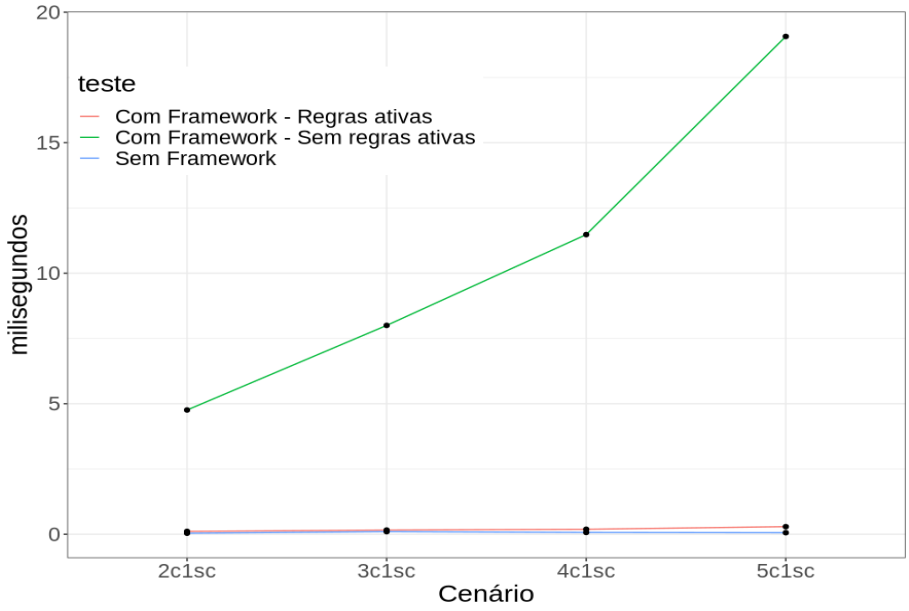
cada FLOWPRI-SDN sempre foi muito baixo em relação ao tempo percebido, principalmente devido a comunicação inicial entre $H1$ e o FLOWPRI-SDN de borda. Nessa comunicação, como o protocolo utilizado é o TCP, deve ocorrer tráfego bilateral. Assim, quando $H1$ vai estabelecer um contrato de QoS com $C1$, existem regras para enviar o tráfego até $C1$, no entanto, não para o tráfego de volta. Deste modo, ocorre um evento *packet_in*, que se cria muito rapidamente (1-3ms) as regras para $C1$ enviar dados para $H1$.

Após a resolução do evento, $H1$ e $C1$ realizam o *handshake* TCP. No entanto, $H1$ demora uma média de 12.12ms para processar os pacotes recebidos, o que acaba por afetar o tempo percebido no estabelecimento do contrato. Além disso, é possível observar que o tempo de comunicação entre *frameworks* para estabelecimento do contrato é em média 2.07ms, obtido da média do tempo consumido entre o primeiro *packet_in* ocorrido em $C1$ até o final do processamento de recebimento de contrato no último *framework* subtraindo o tempo total

de processamento. Portanto, do tempo percebido deve ser considerado que, aproximadamente, 12.12ms são do tempo de comunicação inicial entre $H1$ e $C1$, e 2.07ms são do tempo de comunicação entre *frameworks*, de modo que o restante do tempo é o usado para processamento.

A partir do estabelecimento do contrato de QoS, podem ocorrer duas situações na comunicação entre $H1$ e $H4$: as regras estão ativas nas tabelas de fluxo OVS e o fluxo recebe os recursos exigidos; as regras expiraram nas tabelas de fluxo OVS e é necessário que sejam criadas novamente baseadas no contrato existente por meio do evento *packte_in*. Desta forma, o teste de *overhead* apresentado na Figura 34, busca comparar ambientes com o *framework* vs sem o *framework*, para medir o impacto que o FLOWPRI-SDN possui quando já existe um contrato estabelecido. Neste caso, é verificado o tempo de entrega de um pacote quando as regras estão ativas e quando já expiraram, necessitando criá-las novamente. Neste cenário, é possível observar o tempo que o FLOWPRI-SDN acresce no processamento e no encaminhamento quando comparado com os cenários onde os *switches* já possuem as regras estabelecidas e não precisam de um controlador (simulando roteadores convencionais).

Figura 34 – Comparação de encaminhamento Framework vs Sem Framework.



Fonte: (Próprio Autor, 2022).

Para cada topologia, foram realizadas 10 execuções de coleta de tempo para envio e recebimento do primeiro pacote de um fluxo. Pode se identificar que o tempo de envio de pacotes quando as regras estão ativas nos cenários com FLOWPRI-SDN, mesmo tendo três camadas de encaminhamento entre as tabelas de marcação, encaminhamento e de pré-marcação, o tempo de envio do pacote é praticamente o mesmo de um ambiente pré-configurado com regras utilizando uma camada de tabela de fluxo.

O tempo entre envio e recebimento do pacote quando as regras de fluxo expiraram nas tabelas de fluxo OVS e foram removidas está representada na cor verde. Neste caso, em cada *framework* ocorre o evento *packet_in* que é utilizado para criar as regras novamente nas tabelas

OVS e nas instâncias de *switch* baseadas no contrato já estabelecido, além de iniciar o processo de anúncio de contratos por envio de um pacote ICMP 15. Deste modo, o tempo do cenário sem regras ativas inclui os tempos de processamento e os tempos de comunicação entre *frameworks* no tratamento do protocolo de anúncio e requisição de contrato.

Desta forma, o tempo consumido para recriar as regras necessárias não é tão significativo quanto o tempo de estabelecimento de contratos e só se aplica ao primeiro pacote do fluxos. Portanto, os benefícios do uso do *framework* para reserva de recursos para os fluxos compensa devido ao baixo impacto de *overhead* apresentado.

5.3 CASO DE USO: CIDADE INTELIGENTE

A topologia apresentada na Figura 35 demonstra um cenário de Cidade Inteligente onde um agente policial equipado com uma câmera de vídeo está trabalhando em uma operação importante que precisa ser monitorada e comandada pela central policial a partir do vídeo recebido. O agente está localizado no domínio do controlador 5 e consegue acessar a rede por meio do *switch* OpenFlow do domínio. Deste modo, a câmera envia vídeo em tempo-real para o servidor de nuvem do domínio 1, onde a central policial é capaz de acessar e assistir a operação.

Figura 35 – Cenário Smart City.

Fonte: Próprio Autor, 2022.

No entanto, além de ser usada para aplicações *Smart City*, o servidor da nuvem também fornece outros tipos de dados para aplicações não *Smart City*. Um dispositivo genérico também está conectado no domínio 5 pelo *switch* de acesso. Esse dispositivo envia dados com uma alta carga de tráfego em direção a nuvem. No entanto, esse tráfego não é um tráfego de aplicações

Smart City e acaba por disputar por recursos com a aplicação de monitoramento policial. Deste modo, este tipo de tráfego é chamado de tráfego *background* de baixa prioridade.

Neste cenário, é realizado o teste de funcionalidade, onde é comparado o ambiente com gerenciamento de QoS usando o *framework* vs sem o *framework*. Os enlaces dos *switches* possuem 15Mb de largura de banda disponível, divididos entre as classes de serviço. Nesse caso, a aplicação de *streaming* de vídeo da câmera policial inteligente, como um fluxo UDP em *Full HD*, é enquadrada como aplicação prioritária exigindo largura de banda reservada dada a sua importância e portanto seu tráfego é classificado na classe tempo-real.

Assim, utilizando o *framework*, um contrato é estabelecido entre a câmera inteligente e a nuvem com requisitos conforme os estudos de requisitos de QoS, ou seja, 5$Mbps$ de largura de banda da classe de tráfego de tempo-real, com prioridade 3 (maior) de DSCP 27. A Figura 36, mostra a tabela de fluxos do *switch* $S1$, conectado ao domínio do controlador $C1$, juntamente com a regra *meter* que é criada e utilizada para limitar o uso da largura de banda para o fluxo.

Por outro lado, no cenário sem o *framework*, as regras de encaminhamento são pré-configuradas nos *switches* para que os *hosts* sejam capazes de se comunicar, sem *QoS*. Por fim, nesses dois cenários, a rede não é usada exclusivamente para a câmera e assim um outro dispositivo genérico simula o consumo da largura de banda para um fluxo UDP do tipo *best-effort* (DSCP 60) em direção a nuvem, simulando o tráfego *background* de uma cidade, de dispositivos que não trafegam dados de aplicações *Smart City*. Este fluxo é representado com um servidor *Iperf* no *host* que representa a nuvem no domínio 1 da Figura 35 , e um cliente *Iperf* UDP que tenta usar toda a banda do enlace no domínio 5 da Figura 35.

Figura 36 – Regras criadas com o contrato de QoS no *switch* $S1$.

```
table=1,ip,nw_src=172.16.10.1,nw_dst=172.16.10.4 actions=set_field:7->ip_dscp,goto_table:2
table=1,ip,nw_src=172.16.10.4,nw_dst=172.16.10.1 actions=set_field:60->ip_dscp,goto_table:2
table=2,ip,nw_src=172.16.10.1,nw_dst=172.16.10.4,nw_tos=28 actions=meter:14,set_queue:3,output:4
```

(a) Regras criadas de classificação e encaminhamento.

```
OFPST_METER_CONFIG reply (OF1.3) (xid=0x2):
meter=14 kbps bands=
type=drop rate=5000
```

(b) Regra *meter* criada.

Fonte: (Próprio Autor, 2022).

O comando utilizado para o cliente Iperf é: "iperf -b 15M -u -c 172.16.10.4".

O comando utilizado para o enviar vídeo para a nuvem é: "ffmpeg -re -i video2.mp4 -c:v copy -c:a aac -listen 1 -ar 44100 -preset ultrafast -f mpegts udp://172.16.10.4:10000".

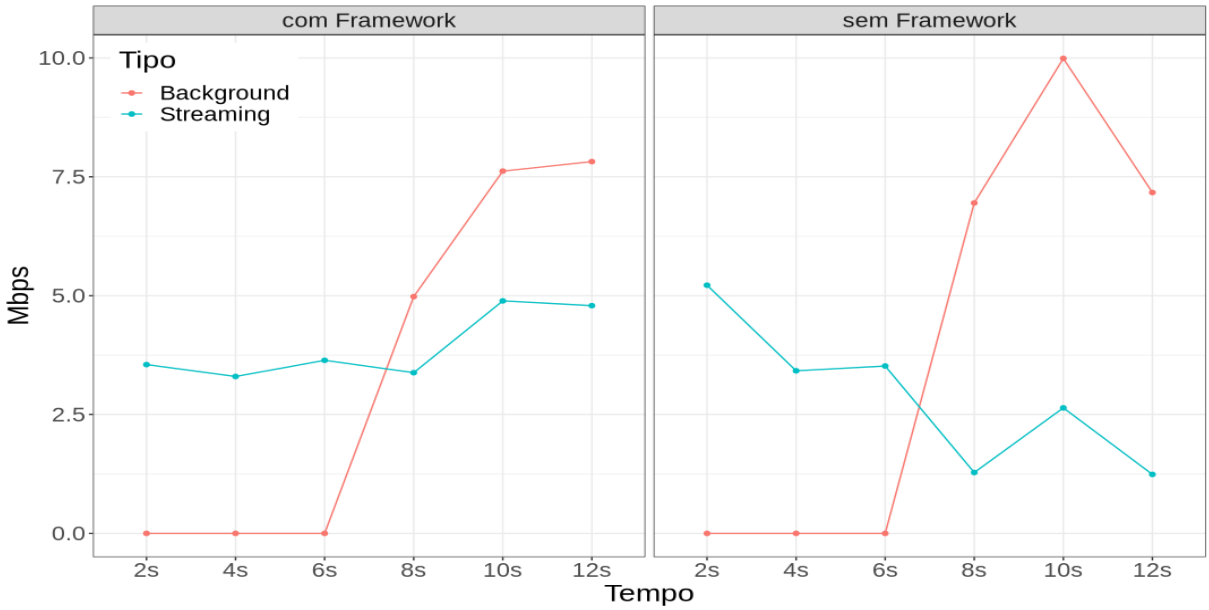
O comando utilizado para receber vídeo no *host* que representa a nuvem é: "mplayer udp://172.16.10.4:10000".

Para monitorar a taxa de recebimento de dados de cada fluxo na nuvem durante o recebimento de tráfego de vídeo e de tráfego *background*, foi utilizada a ferramenta Iftop. Essa ferramenta pode ser instalada facilmente e é capaz de monitorar a largura de banda ativa de cada interface (CARRIGAN, 2020).

A Figura 37 apresenta o comportamento do tráfego de *streaming* de vídeo prioritário em um cenário com tráfego *background* (*best-effort*) intenso. Como se pode perceber, no caso em que é usado o FLOWPRI-SDN para gerenciar cada domínio, o fluxo prioritário tem a largura de banda reservada e não é influenciada por um fluxo de menor prioridade, como o tráfego *background*, representado pelo *Iperf*. Neste caso, a taxa de recebimento se mantém constante, mesmo quando o tráfego *background* tenta utilizar toda a largura de banda do enlace.

No entanto, em um cenário sem o uso do *framework*, a aplicação prioritária fica bastante comprometida quando o tráfego *Iperf* se torna ativo. Neste caso, o comportamento de disputa de largura de banda começa a ser evidente, quando o tráfego *background* e da aplicação de vídeo oscilam, por momentos a aplicação de vídeo utiliza a largura de banda desejada e em outros momentos, o tráfego *background* utiliza largura de banda desejada.

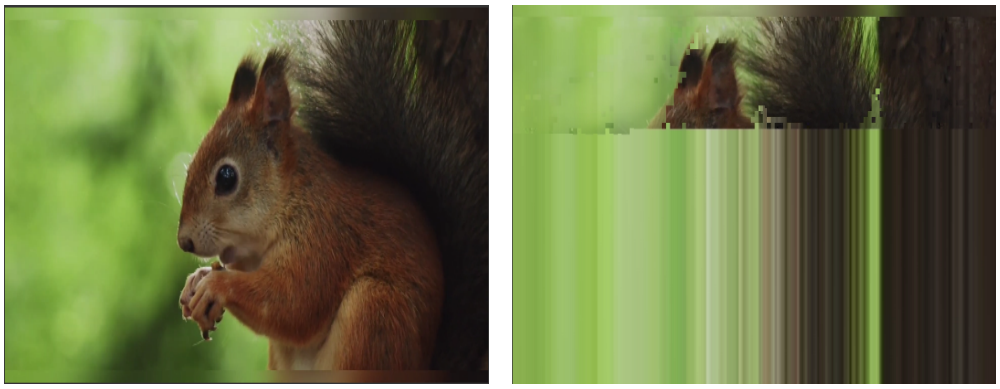
Figura 37 – Teste de largura de banda.



Fonte: (Próprio Autor, 2022).

Como resultado, a qualidade do vídeo recebido sofre muito com o tráfego sem priorização. A Figura 38, valida o uso do *framework* quando demonstra a perda da qualidade dos *frames* recebidos, fazendo com que uma aplicação prioritária falhasse em um cenário específico, como em uma cidade inteligente.

Figura 38 – Streaming com tráfego background.



(a) Cenário com FLOWPRI-SDN. (b) Cenário sem FLOWPRI-SDN.
Fonte: (Próprio Autor, 2022).

5.4 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram apresentadas questões de implementação do FLOWPRI-SDN para um ambiente de simulação Mininet, quando é necessário simular domínios multi-controladores em uma mesma máquina. A solução utilizada foi baseada em traduções de endereços, de forma que cada controlador tenha um endereço fictício específico para se comunicar com outro, marcado na tabela de roteamento para encaminhar tráfego pela interface de rede correta.

Foram apresentados também, os testes referentes a sobrecarga de comunicação que o FLOWPRI-SDN adiciona para que fluxos trafeguem com o QoS solicitado. Para o primeiro teste, foi medido o tempo envolvido no estabelecimento do contrato de QoS em quatro topologias diferentes. No segundo teste foi observado o tempo que se leva para recriar regras de fluxo quando um contrato já está estabelecido. Nesses testes, foi possível identificar que o tempo percebido na criação do contrato inclui também o tempo que o emissor do contrato leva para se comunicar com o FLOWPRI-SDN, que representa uma porção considerável desse tempo. Desta forma, nos testes se pode perceber que o tempo que envolve apenas o processamento nos *frameworks* demonstra que o *framework* pode ser uma boa opção para ambientes dinâmicos de solicitação de recursos.

No teste de aplicação, o FLOWPRI-SDN foi testado em um cenário de cidade inteligente. Um dispositivo prioritário representado pela câmera de vídeo inteligente busca enviar tráfego de vídeo em direção à nuvem. Juntamente com o fluxo de vídeo, foi simulado um tráfego *background* para testar o comportamento de um cenário com FLOWPRI-SDN e sem. Desta forma, foi demostrado que com o *framework* fluxos prioritários com recursos reservados trafegam com maior qualidade que em cenários sem recursos reservados.

6 CONCLUSÃO

O trabalho desenvolvido, buscou abordar os problemas de fornecimento de QoS e priorização para fluxos que necessitam de reserva de recursos, na forma do *framework* FLOWPRI-SDN. Este *framework* pode ser aplicado a ambientes com necessidades dinâmicas de QoS, como o caso de cenários de cidades inteligentes.

Para compreender os processos de dados de uma cidade inteligente, realizou-se o estudo do conceito de cidades inteligentes, bem como suas arquiteturas e aplicações. Além disso, foram estudados os principais dispositivos geradores de dados e os meios de transmissão de dados utilizados para comunicação entre os dispositivos e os serviços de nuvem utilizados para processamento de dados das cidades.

Com isso, identificou-se que o gerenciamento dos recursos de rede em redes convencionais é muito complexa devido a questões da arquitetura. Assim, o uso de redes baseadas na arquitetura SDN foi abordado, pois permitem programar os elementos de rede de forma dinâmica e sua implementação deve se tornar cada vez mais comum em redes da Internet.

A revisão bibliográfica dos conceitos de QoS, possibilitou a identificação dos mecanismos que podem auxiliar no desenvolvimento de ferramentas para fornecer largura de banda e priorização de forma dinâmica e conforme a necessidade dos fluxos de dados. Além disso, realizou-se uma pesquisa sobre os trabalhos relacionados, que permitiram identificar as soluções existentes que relacionam QoS, priorização, SDN e cidades inteligentes, validando a proposta deste trabalho.

Com base nessas pesquisas foi proposto o *framework* FLOWPRI-SDN, para gerenciar os recursos de rede e fornecer largura de banda e priorização para os fluxos de dados que identificarem essa necessidade, em cidades inteligentes. Assim, o *framework* possui uma arquitetura baseada, principalmente, em ações como: enviar e receber contratos de QoS, realizar anúncio e solicitação de contratos e alocar fluxos com reserva de recurso e sem reserva de recurso.

Os resultados apresentados demonstram que o FLOWPRI-SDN pode ser utilizada em cenários dinâmicos e que os fluxos recebem a largura de banda solicitada de forma reservada. Além disso, o tempo consumido nos processos internos do *framework* não são um impeditivo para sua aplicação.

Para trabalhos futuros, sugere-se expandir a implementação do FLOWPRI-SDN para que os contratos suportem configuração de portas e suportar múltiplos fluxos entre *hosts* cada um com requisitos diferentes. Além disso, melhorar os componentes de reserva de recursos para suportar outras opções de largura de banda, além dos apresentados pelos códigos DSCP. Sugere-se também, promover uma forma de identificação de fluxos sem solicitar ao emissor, bem como, criar um mecanismo para identificar dispositivos e fluxos de cidade inteligente para restringir fluxos de alta prioridade aos dispositivos de cidade inteligente autorizados.

REFERÊNCIAS

AAZAM, Mohammad et al. Cloud-based smart waste management for smart cities. In: **2016 IEEE 21st International Workshop on Computer Aided Modelling and Design of Communication Links and Networks (CAMAD)**. [S.l.: s.n.], 2016. p. 188–193. Citado na página 94.

ACAMPORA, Giovanni et al. A survey on ambient intelligence in healthcare. **Proceedings of the IEEE**, v. 101, n. 12, p. 2470–2494, 2013. Citado na página 94.

AGLIANò, Simone et al. Resource management and control in virtualized sdn networks. In: **2018 Real-Time and Embedded Systems and Technologies (RTEST)**. [S.l.: s.n.], 2018. p. 47–53. Citado 3 vezes nas páginas 48, 49 e 50.

AL-FUQAHA, Ala et al. Toward better horizontal integration among iot services. **IEEE Communications Magazine**, v. 53, n. 9, p. 72–79, 2015. Citado na página 20.

AL-HADDAD, Ronak et al. A novel traffic shaping algorithm for sdn-sliced networks using a new wfq technique. **International Journal of Advanced Computer Science and Applications**, The Science and Information Organization, v. 12, n. 1, 2021. Disponível em: <http://dx.doi.org/10.14569/IJACSA.2021.0120101>. Citado na página 32.

ALAM, Iqbal et al. A survey of network virtualization techniques for internet of things using sdn and nfv. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 2, abr. 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3379444>. Citado na página 13.

ALAVI, Amir H et al. Internet of things-enabled smart cities: State-of-the-art and future trends. **Measurement**, Elsevier, v. 129, p. 589–606, 2018. Citado na página 23.

ALHARBI, N; SOH, B. Roles and challenges of network sensors in smart cities. In: IOP PUBLISHING. **IOP Conference Series: Earth and Environmental Science**. [S.l.], 2019. v. 322, n. 1, p. 012002. Citado na página 19.

ALLIANCE, NGMN. 5g white paper. **Next generation mobile networks, white paper**, v. 1, n. 2015, 2015. Citado na página 15.

ANDERSON, Mike. **Networking the IoT with IEEE 802.15.4/6LoWPAN**. [S.l.], 2017. Disponível em: <https://www.techbriefs.com/component/content/article/tb/supplements/st/features/articles/27510>. Citado na página 22.

AUTHORITY, Rwanda Utilities Regulatory. **GUIDELINE N 01/GL/UAS-ICS/ RURA/018 OF 07/06/2018 ON MINIMUM BANDWIDTH FOR BROADBAND INTERNET CONNECTIVITY IN RWANDA**. [S.l.], 2018. Citado 2 vezes nas páginas 13 e 26.

BABIARZ K. CHAN, F. Baker J. **Configuration Guidelines for DiffServ Service Classes**. [S.l.], 2006. Disponível em: <https://datatracker.ietf.org/doc/html/rfc4594>. Citado na página 31.

BARIK, Runa et al. On the utility of unregulated ip diffserv code point (dscp) usage by end systems. **Performance Evaluation**, Elsevier, v. 135, p. 102036, 2019. Citado na página 31.

Belgaum, M. R. et al. Integration challenges of artificial intelligence in cloud computing, internet of things and software-defined networking. In: **2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)**. [S.l.: s.n.], 2019. p. 1–5. Citado na página 37.

BERNTZEN, Lasse; JOHANNESSEN, Marius Rohde; FLOREA, Adrian. Smart cities: Challenges and a sensor-based solution a research design for sensor-based smart city projects. 2016. Citado na página 20.

BHOLEBAWA, Idris; DALAL, Upena. Design and performance analysis of openflow-enabled network topologies using mininet. **International Journal of Computer and Communication Engineering**, v. 5, p. 419–429, 01 2016. Citado na página 43.

CARRIGAN, Tyler. **Linux interface analytics on-demand with iftop**. 2020. <https://www.redhat.com/sysadmin/linux-interface-iftop>. Citado na página 82.

CHEN, Yan; FARLEY, Toni; YE, Nong. Qos requirements of network applications on the internet. **Systems Management**, v. 4, 01 2004. Citado 3 vezes nas páginas 13, 25 e 26.

CISCO. **Implementando políticas de qualidade do serviço com DSCP**. [S.l.], 2008. Citado 2 vezes nas páginas 30 e 52.

CISCO. **Chapter: DSCP and Precedence Values**. 2016. <https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus1000/sw/4_0_4_s_v_1_3/qos/configuration/guide/n1000v_qos/n1000v_qos_6dscpval.html>. Citado na página 32.

CISCO. **What Is a Smart City?** c2022. <https://www.cisco.com/c/en/us/solutions/industries/smart-connected-communities/what-is-a-smart-city.html>. Citado na página 16.

COMERCIAL, Mult. **TIPOS DE SENSORES: SAIBA QUAIS SÃO E AS APLICAÇÕES DE CADA UM**. 2020. Urlhttps://blog.multcomercial.com.br/tipos-de-sensores/. Citado na página 19.

COSTA, Daniel et al. Research trends in wireless visual sensor networks when exploiting prioritization. **Sensors**, v. 15, p. 1760–1784, 01 2015. Citado 4 vezes nas páginas 12, 26, 27 e 28.

COSTA, Daniel G; OLIVEIRA, Felipe P de. A prioritization approach for optimization of multiple concurrent sensing applications in smart cities. **Future Generation Computer Systems**, Elsevier, v. 108, p. 228–243, 2020. Citado 8 vezes nas páginas 12, 19, 22, 26, 27, 28, 49 e 61.

CRISTIANE. **Protocolo MPLS: Estudo sobre aspectos funcionais**. [S.l.], 2014. Disponível em: <https://www.devmedia.com.br/protocolo-mpls-estudo-sobre-aspectos-funcionais/30652>. Citado na página 29.

DECUIR, Joseph. Introducing bluetooth smart: Part 1: A look at both classic and new technologies. **IEEE Consumer Electronics Magazine**, v. 3, n. 1, p. 12–18, 2014. Citado na página 22.

DEERING, S.; HINDEN, R. **Internet Protocol, Version 6 (IPv6) Specification**. [S.l.], 1998. Disponível em: <https://datatracker.ietf.org/doc/html/rfc2460>. Citado na página 30.

DIORIO, Rafael; TIMóTEO, Varese. Per-flow routing with qos support to enhance multimedia delivery in openflow sdn. In: . [S.l.: s.n.], 2016. p. 167–174. Citado 2 vezes nas páginas 48 e 50.

EGILMEZ, Hilmi; TEKALP, A. Distributed qos architectures for multimedia streaming over software defined networks. **Multimedia, IEEE Transactions on**, v. 16, p. 1597–1609, 10 2014. Citado 3 vezes nas páginas 48, 49 e 50.

ESPOSTE, Arthur de M Del et al. Design and evaluation of a scalable smart city software platform with large-scale simulations. **Future Generation Computer Systems**, Elsevier, v. 93, p. 427–441, 2019. Citado na página 18.

FENG, Daquan et al. Device-to-device communications in cellular networks. **IEEE Communications Magazine**, v. 52, n. 4, p. 49–55, 2014. Citado na página 22.

FINKENZELLER, Klaus. **RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication**. [S.l.]: John wiley & sons, 2010. Citado na página 19.

GHARAIBEH, Ammar et al. Smart cities: A survey on data management, security, and enabling technologies. **IEEE Communications Surveys & Tutorials**, IEEE, v. 19, n. 4, p. 2456–2501, 2017. Citado 3 vezes nas páginas 12, 15 e 18.

GONT, C. Pignataro F. **Formally Deprecating Some ICMPv4 Message Types**. [S.l.], 2013. Disponível em: <https://datatracker.ietf.org/doc/html/rfc6918>. Citado na página 55.

GULERIA, Ajay. Traffic engineering in software defined networks: A survey. **Journal of Telecommunications and Information Technology**, v. 4, p. 3–14, 12 2016. Citado na página 12.

HARYADI, Sigit. **Telecommunication Service and Experience Quality**. [S.l.: s.n.], 2013. ISBN 978-602-18578-6-1. Citado na página 25.

Hu, N. et al. A novel sdn-based application-awareness mechanism by using deep learning. **IEEE Access**, v. 8, p. 160921–160930, 2020. Citado na página 13.

HUANG, Chen-Hao; CHOU, Tzu-Chuan; WU, Sheng-Hsiung. Towards convergence of ai and iot for smart policing: a case of a mobile edge computing-based context-aware system. **Journal of Global Information Management (JGIM)**, IGI Global, v. 29, n. 6, p. 1–21, 2021. Citado na página 24.

IPV6.BR. **Cabeçalho**. 2022. <https://ipv6.br/post/cabecalho/>. Citado na página 30.

ITU. **End-user multimedia QoS categories**. [S.l.], 2001. Citado 2 vezes nas páginas 25 e 26.

ITU, TELECOMMUNICATION STANDARDIZATION SECTOR OF. **SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS Quality of service and performance**. [S.l.], 2001. Citado na página 13.

JANEVSKI, Toni; JANKOVIC, Milan; MARKUS, Scott. **Quality of service regulation manual**. [S.l.], 2017. Citado 2 vezes nas páginas 25 e 53.

JANG, Hung-Chin; LIN, Jian-Ting. Sdn based qos aware bandwidth management framework of isp for smart homes. In: **2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)**. [S.l.: s.n.], 2017. p. 1–6. Citado 2 vezes nas páginas 47 e 50.

JIN, Li-jie; MACHIRAJU, Vijay; SAHAI, Akhil. Analysis on service level agreement of web services. **HP June**, v. 19, p. 1–13, 2002. Citado na página 29.

KARAKUS, Murat; DURRESI, Arjan. Quality of service (qos) in software defined networking (sdn): A survey. **Journal of Network and Computer Applications**, v. 80, p. 200 – 218, 2017. ISSN 1084-8045. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1084804516303186>. Citado 3 vezes nas páginas 25, 37 e 39.

KASHEF, Mohamad; VISVIZI, Anna; TROISI, Orlando. Smart city as a smart service system: Human-computer interaction and smart city surveillance systems. **Computers in Human Behavior**, v. 124, p. 106923, 2021. ISSN 0747-5632. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0747563221002466>. Citado na página 24.

KAUR, Karamjeet; SINGH, Japinder; GHUMMAN, Navtej. Mininet as software defined networking testing platform. In: . [S.l.: s.n.], 2014. Citado na página 71.

KERRISK, Michael. **The Linux programming interface: a Linux and UNIX system programming handbook**. [S.l.]: No Starch Press, 2010. Citado 3 vezes nas páginas 34, 35 e 36.

KHANVILKAR, Shashank et al. **Multimedia networks and communication**. [S.l.]: Citeseer, 2004. 431–432 p. Citado na página 25.

Krainyk, Y.; Dvornik, O.; Krainyk, O. Software-defined network application-aware controller for internet-of-things. In: **2019 3rd International Conference on Advanced Information and Communications Technologies (AICT)**. [S.l.: s.n.], 2019. p. 165–169. Citado na página 37.

KRISHNA, Hedi; ADRICHEM, Niels L. M. van; KUIPERS, Fernando A. Providing bandwidth guarantees with openflow. In: **2016 Symposium on Communications and Vehicular Technologies (SCVT)**. [S.l.: s.n.], 2016. p. 1–6. Citado 3 vezes nas páginas 36, 48 e 50.

KROODO, Märt. **2G / 3G / 4G / 5G / NB-IoT / LTE-M – Which to Choose for Your IoT Project?** [S.l.], 2019. Disponível em: <https://1ot.mobi/resources/blog/gsm-3g-4g-lte-which-one-to-choose-for-your-iot-project>. Citado na página 22.

KRUGER, Lennard G. **Defining broadband: minimum threshold speeds and broadband policy**. [S.l.]: Congressional Research Service Washington, DC, 2017. Citado 3 vezes nas páginas 13, 25 e 26.

LEONARDI, Luca; BELLO, Lucia Lo; AGLIANò, Simone. Priority-based bandwidth management in virtualized software-defined networks. **Electronics**, v. 9, p. 1009, 06 2020. Citado 2 vezes nas páginas 47 e 50.

LIBELIUM. **50 Sensor Applications for a Smarter World**. 2020. <https://www.libelium.com/libeliumworld/top_50_iot_sensor_applications_ranking/#Smart_cities>. Citado na página 94.

LÓPEZ-QUILES, José Miguel; BOLÍVAR, Manuel Pedro Rodríguez. Smart technologies for smart governments: A review of technological tools in smart cities. **Smart technologies for smart governments**, Springer, p. 1–18, 2018. Citado na página 94.

MAINUDDIN, Md; DUAN, Zhenhai; DONG, Yingfei. Network traffic characteristics of iot devices in smart homes. In: IEEE. **2021 International Conference on Computer Communications and Networks (ICCCN)**. [S.l.], 2021. p. 1–11. Citado 2 vezes nas páginas 13 e 26.

MAKAM, Sreenivas. **Openvswitch and ovsdb**. [S.l.], 2014. Disponível em: <https://sreeninet.wordpress.com/2014/01/02/openvswitch-and-ovsdb/>. Citado na página 41.

MARTINS, Joberto; REALE, Rafael; BEZERRA, Romildo Martins da Silva. Analysis of bandwidth allocation models reconfiguration impacts. In: . [S.l.: s.n.], 2014. Citado 2 vezes nas páginas 33 e 34.

MARTINS, Joberto; REALE, Rafael; BEZERRA, Romildo Martins da Silva. G-bam: A generalized bandwidth allocation model for ip/mpls/ds-te networks. v. 6, p. 635–643, 12 2014. Citado na página 33.

MOCNEJ, Jozef et al. **Network traffic characteristics of the IoT application use cases**. [S.l.]: School of Engineering and Computer Science, Victoria University of . . . , 2018. Citado 4 vezes nas páginas 13, 26, 27 e 28.

MOVASSAGHI, Samaneh et al. Wireless body area networks: A survey. **IEEE Communications Surveys & Tutorials**, v. 16, n. 3, p. 1658–1686, 2014. Citado na página 94.

MYEONG, Seunghwan; JUNG, Yuseok; LEE, Eunuk. A study on determinant factors in smart city development: An analytic hierarchy process analysis. **Sustainability**, v. 10, n. 8, 2018. ISSN 2071-1050. Disponível em: <https://www.mdpi.com/2071-1050/10/8/2606>. Citado na página 12.

NDIAYE, Musa; HANCKE, Gerhard P.; ABU-MAHFOUZ, Adnan M. Software defined networking for improved wireless sensor network management: A survey. **Sensors**, v. 17, n. 5, 2017. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/17/5/1031>. Citado na página 13.

NGUYEN-HOANG, Phuc; VO-TAN, Phuoc. Development an open-source industrial iot gateway. In: **2019 19th International Symposium on Communications and Information Technologies (ISCIT)**. [S.l.: s.n.], 2019. p. 201–204. Citado na página 22.

NICHOLS, K. et al. **Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers**. [S.l.], 1998. Disponível em: <https://datatracker.ietf.org/doc/html/rfc2474>. Citado 2 vezes nas páginas 30 e 54.

NIKOUEI, Seyed Yahya; CHEN, Yu; FAUGHNAN, Timothy R. Smart surveillance as an edge service for real-time human detection and tracking. In: **2018 IEEE/ACM Symposium on Edge Computing (SEC)**. [S.l.: s.n.], 2018. p. 336–337. Citado na página 24.

OGRODOWCZYK, Łukasz; BELTER, Bartosz; LECLERC, Marc. Iot ecosystem over programmable sdn infrastructure for smart city applications. In: IEEE. **2016 Fifth European Workshop on Software-Defined Networks (EWSDN)**. [S.l.], 2016. p. 49–51. Citado na página 18.

OKAI, Ebenezer; FENG, Xiaohua; SANT, Paul. Smart cities survey. In: IEEE. **2018 IEEE 20th international conference on high performance computing and communications; IEEE 16th international conference on smart city; IEEE 4th international conference on data science and systems (HPCC/SmartCity/DSS)**. [S.l.], 2018. p. 1726–1730. Citado na página 15.

OLIVEIRA, Rogerio et al. Using mininet for emulation and prototyping software-defined networks. In: . [S.l.: s.n.], 2014. p. 1–6. ISBN 978-1-4799-4340-1. Citado na página 71.

ONF. **OpenFlow Switch Specification Version 1.3**. [S.l.], 2012. Citado 3 vezes nas páginas 38, 44 e 45.

ONF. **Software-Defined Networking: The New Norm for Networks**. [S.l.], 2012. Disponível em: <https://pdfs.semanticscholar.org/a3f6/9f6181a0b4d481073a21eafbcc434a800db6.pdf>. Citado 2 vezes nas páginas 37 e 39.

(ONF), Open Networking Foundation. **Software-Defined Networking: The New Norm for Networks**. [S.l.], 2012. Citado na página 42.

PALARMINI, Luiz Fernando. **Conheça os principais protocolos para IoT**. [S.l.], 2020. Disponível em: <https://www.filipeflop.com/blog/conheca-os-principais-protocolos-para-iot/>. Citado 2 vezes nas páginas 20 e 23.

Paliwal, M.; Shrimankar, D.; Tembhurne, O. Controllers in sdn: A review report. **IEEE Access**, v. 6, p. 36256–36270, 2018. Citado 2 vezes nas páginas 38 e 39.

PEREIRA, Gabriela et al. Smart governance in the context of smart cities: A literature review. **Information Polity**, v. 23, p. 1–20, 05 2018. Citado na página 94.

PETROLO, Riccardo; LOSCRI, Valeria; MITTON, Nathalie. Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms. **Transactions on emerging telecommunications technologies**, Wiley Online Library, v. 28, n. 1, p. e2931, 2017. Citado na página 19.

PFAFF, B. Davie B. **The Open vSwitch Database Management Protocol**. [S.l.], 2013. Disponível em: <https://datatracker.ietf.org/doc/html/rfc7047.txt>. Citado na página 41.

POSTEL, J. **INTERNET CONTROL MESSAGE PROTOCOL**. [S.l.], 1981. Disponível em: <https://datatracker.ietf.org/doc/html/rfc792>. Citado 2 vezes nas páginas 54 e 56.

RAMÍREZ-MORENO, Mauricio A et al. Sensors for sustainable smart cities: A review. **Applied Sciences**, Multidisciplinary Digital Publishing Institute, v. 11, n. 17, p. 8198, 2021. Citado 2 vezes nas páginas 15 e 19.

REN, Shuangyin. A service curve of hierarchical token bucket queue discipline on soft-ware defined networks based on deterministic network calculus: An analysis and simulation. **Journal of Advances in Computer Networks**, p. 8–12, 2017. Citado na página 35.

ROSEN A. VISWANATHAN, R. Callon E. **Multiprotocol Label Switching Architecture**. [S.l.], 2001. Disponível em: <https://datatracker.ietf.org/doc/html/rfc3031>. Citado na página 29.

RUTRONIK. **IEEE 802.11ah alias WiFi HaLow - The best of WiFi and LPWAN**. [S.l.], 2021. Disponível em: <https://www.rutronik.com/article/ieee-80211ah-alias-wifi-halow-the-best-of-wifi-and-lpwan/>. Citado na página 22.

SANTANA, Eduardo Felipe Zambom et al. Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. **ACM Computing Surveys (Csur)**, ACM New York, NY, USA, v. 50, n. 6, p. 1–37, 2017. Citado 4 vezes nas páginas 16, 18, 22 e 23.

SANTOS, José et al. Towards end-to-end resource provisioning in fog computing over low power wide area networks. **Journal of Network and Computer Applications**, v. 175, p. 102915, 2021. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S108480452030374X>. Citado na página 20.

SEDDIKI, Mohamed Said et al. Flowqos: Per-flow quality of service for broadband access networks. In: . [S.l.: s.n.], 2015. Citado 3 vezes nas páginas 48, 49 e 50.

Selvaraj, P.; Nagarajan, V. Migration from conventional networking to software defined networking. In: **2017 International Conference on IoT and Application (ICIOT)**. [S.l.: s.n.], 2017. p. 1–7. Citado 2 vezes nas páginas 39 e 41.

SINAEEPOURFARD, Amir et al. A data lifecycle model for smart cities. In: IEEE. **2016 international conference on information and communication technology convergence (ICTC)**. [S.l.], 2016. p. 400–405. Citado na página 18.

SINAEEPOURFARD, Amir; KROGSTIE, John; PETERSEN, Sobah Abbas. A big data management architecture for smart cities based on fog-to-cloud data management architecture. CEUR WS Aachen University, 2018. Citado na página 18.

SOBYTE. **Open vSwitch**. [S.l.], 2022. Disponível em: <https://www.sobyte.net/post/2022-04/open-vswitch/>. Citado na página 40.

SOPHOS. **DSCP Value**. c2018. <https://docs.sophos.com/nsg/sophos-firewallmanager/v17.0.0/Help/en-us/webhelp/onlinehelp/index.html#page/onlinehelp/DSCPValue.html>. Citado na página 32.

SYED, Abbas Shah et al. Iot in smart cities: a survey of technologies, practices and challenges. **Smart Cities**, Multidisciplinary Digital Publishing Institute, v. 4, n. 2, p. 429–475, 2021. Citado na página 17.

TADINADA, Vishwapathi Rao. Software defined networking: Redefining the future of internet in iot and cloud era. In: **2014 International Conference on Future Internet of Things and Cloud**. [S.l.: s.n.], 2014. p. 296–301. Citado na página 22.

TEAM, RYU project. **RYU SDN Framework, Release 1.0**. [S.l.], 2014. Citado na página 38.

TOMOVIC, Slavica; PRASAD, Neeli; RADUSINOVIC, Igor. Sdn control framework for qos provisioning. In: **2014 22nd Telecommunications Forum Telfor (TELFOR)**. [S.l.: s.n.], 2014. p. 111–114. Citado 2 vezes nas páginas 48 e 50.

TORRES, Eliseu et al. A SDN/OpenFlow Framework for Dynamic Resource Allocation based on Bandwidth Allocation Model. **IEEE Latin America Transactions**, v. 18, n. 5, p. 853–860, abr. 2020. Disponível em: <https://doi.org/10.5281/zenodo.3766178>. Citado 3 vezes nas páginas 33, 47 e 50.

TSCHOFENIG, H.; ARKKO, J. **Report from the Smart Object Workshop**. [S.l.], 2012. Disponível em: <https://datatracker.ietf.org/doc/html/rfc6574>. Citado 2 vezes nas páginas 20 e 21.

TSCHOFENIG, H. et al. **Architectural Considerations in Smart Object Networking**. [S.l.], 2015. Disponível em: <https://datatracker.ietf.org/doc/html/rfc7452>. Citado 2 vezes nas páginas 12 e 21.

UFSC. **Biossensores**. 2022. <https://loosa.paginas.ufsc.br/biossensores/>. Citado na página 19.

USTEV, Yunus Emre; INCEL, Ozlem Durmaz; ERSOY, Cem. User, device and orientation independent human activity recognition on mobile phones: Challenges and a proposal. In: **Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication**. [S.l.: s.n.], 2013. p. 1427–1436. Citado na página 20.

VAILSHERY, Lionel Sujay. **"Internet of Things (IoT) and non-IoT active device connections worldwide from 2010 to 2025**. [S.l.], 2021. Disponível em: <https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/>. Citado na página 24.

VPNMENTOR. **The Ultimate Guide to VPN Tunneling & How To Use It In 2022**. 2022. <https://www.vpnmentor.com/blog/ultimate-guide-to-vpn-tunneling/>. Citado na página 32.

WAY, Admiralty. **INTERNET PROTOCOL**. [S.l.], 1981. Disponível em: <https://datatracker.ietf.org/doc/html/rfc791>. Citado na página 30.

WENGE, Rong et al. Smart city architecture: A technology guide for implementation and design challenges. **China Communications**, IEEE, v. 11, n. 3, p. 56–69, 2014. Citado 2 vezes nas páginas 12 e 18.

YAN, Jinyao et al. Hiqos: An sdn-based multipath qos solution. **China Communications**, v. 12, n. 5, p. 123–133, 2015. Citado 2 vezes nas páginas 47 e 50.

YIN, Chuantao et al. A literature survey on smart cities. **Science China Information Sciences**, v. 58, 08 2015. Citado 5 vezes nas páginas 12, 17, 23, 24 e 94.

ZHANG, Changhao. Design and application of fog computing and internet of things service platform for smart city. **Future Generation Computer Systems**, Elsevier, v. 112, p. 630–640, 2020. Citado na página 18.

ZHANG, Kuan et al. Security and privacy in smart city applications: Challenges and solutions. **IEEE Communications Magazine**, v. 55, n. 1, p. 122–129, 2017. Citado na página 15.

Zhang, Z. et al. An sdn-based network architecture for internet of things. In: **2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)**. [S.l.: s.n.], 2018. p. 980–985. Citado 2 vezes nas páginas 13 e 23.

ZUBIZARRETA, Iker; SERAVALLI, Alessandro; ARRIZABALAGA, Saioa. Smart city concept: What it is and what it should be. **Journal of Urban Planning and Development**, American Society of Civil Engineers, v. 142, n. 1, p. 04015005, 2016. Citado na página 16.

ANEXO A – DOMÍNIOS, SUBDOMÍNIOS E APLICAÇÕES RELACIONADAS A CIDADES INTELIGENTES.

Domínio (Principal objetivo)	Subdomínio	Aplicações relacionadas
Governo (Eficiência)	*E-governo *Governo transparente *Serviços públicos *Segurança pública *Monitoramento da cidade *Resposta à emergências	1-Compartilhamento de informação entre departamentos públicos da administração (PEREIRA et al., 2018). 2-Site específico para fornecer *e-services* no geral (LÓPEZ-QUILES; BOLÍVAR, 2018). 3-Plataforma para participação dos cidadãos em *feedbacks* (LÓPEZ-QUILES; BOLÍVAR, 2018). 4-Distribuição eficiente de efetivo policial pela cidade. 5-Reconhecimento de imagens e rastreamento inteligente de suspeitos e criminosos.
Cidadãos (Felicidade)	*Transporte público *Tráfego inteligente *Turismo *Entretenimento *Assistência médica *Educação *Coesão social	1-Detecção de problemas na via, como obras e buracos, por meio de sensores instalados nas vias e por *feedback* (LIBELIUM, 2020). 2-Detecção de problemas nos veículos e localização. Sensores instalados na via informam sobre faróis não funcionando, pneu vazio e entre outros. 3-Gerenciamento dinâmico dos semáforos. 4-Gerenciamento inteligente da energia elétrica.
Negócios (Lucro)	*Gestão empresarial *Logística *Cadeias de abastecimento *Transações *Propaganda *Inovação *Empreendedorismo *Agricultura	1-Dispositivos implantados no corpo que permitam o seu monitoramento podem prevenir problemas como o infarto do miocárdio e outras condições anormais. Informando periodicamente os médicos responsáveis por meio de uma plataforma (MOVASSAGHI et al., 2014). 2-Controle remoto de dispositivos médicos permite a rede de dispositivos e serviços em atendimento domiciliar, com objetivo de tratar pacientes que são atendidos em seu domicílio, o que diminui os custos para a sociedade (MOVASSAGHI et al., 2014). 3-Sistemas ubíquos baseados em *insights* neurológicos e psicológicos podem ser usados para analisar emoções e melhorar o bem-estar das pessoas (ACAMPORA et al., 2013). 4-Monitoramento da saúde animal usando sensores para coletar informações como temperatura e pressão sanguínea. 5-Rastreamento de animais, implantando dispositivos que transmitem a localização de GPS para uma aplicação.
Ambiente (Sustentabilidade)	**Smart grid* *Energia renovável *Gerenciamento de água *Gerenciamento de resíduos *Controle da poluição *Construção *Habitação *Comunidade *Espaço público	1-Medição da pressão da água em sistemas de transporte de água, ajuda a prevenir e encontrar problemas na distribuição, como vazamentos (LIBELIUM, 2020). 2-Monitorar a qualidade da água da torneira nas cidades (LIBELIUM, 2020). 3-Monitoramento da umidade do solo, vibrações e densidade do solo para detectar padrões perigosos nas condições do solo e evitar problemas como deslizamentos (LIBELIUM, 2020). 4-Dispositivos que monitorem o desperdício de comida em lixeiras orgânicas, para gerar multa e evitar o desperdício (AAZAM et al., 2016). 5-Monitorização de vibrações e condições materiais em edifícios, pontes e monumentos históricos, para verificar a saúde estrutural e criar estratégias de manutenção, por exemplo (LIBELIUM, 2020).

Fonte: Adaptado de (YIN et al., 2015)