UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

EDWARDES AMARO GALHARDO

Implementação e Avaliação de Múltiplas Tabelas de Fluxo em *Switches OpenFlow* Aplicadas em vCPE e SD-VANETs

Goiânia
2020

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO (TECA) PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES

E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo das Teses e Dissertações disponibilizado na BDTD/UFG é de responsabilidade exclusiva do autor. Ao encaminhar o produto final, o autor(a) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do material bibliográfico

[x] Dissertação [ ] Tese

2. Nome completo do autor

Edwardes Amaro Galhardo

3. Título do trabalho

Implementação e Avaliação de Múltiplas Tabelas de Fluxo em Switches OpenFlow Aplicadas em vCPE e SD-VANETs

4. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador)

Concorda com a liberação total do documento [x] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante:
a) consulta ao(à) autor(a) e ao(à) orientador(a);
b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo da tese ou dissertação.
O documento não será disponibilizado durante o período de embargo.
Casos de embargo:
- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro;
- Publicação da dissertação/tese em livro.

Obs. Este termo deverá ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Antonio Carlos De Oliveira Junior**, **Professor do Magistério Superior**, em 03/11/2020, às 08:58, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **EDWARDES AMARO GALHARDO**, **Discente**, em 03/11/2020, às 14:06, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1650619** e o código CRC **EB817356**.

Referência: Processo nº 23070.042302/2020-89 SEI nº 1650619

EDWARDES AMARO GALHARDO

Implementação e Avaliação de Múltiplas Tabelas de Fluxo em *Switches OpenFlow* Aplicadas em vCPE e SD-VANETs

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Sistemas de Computação - Redes de Computadores.

Orientador: Prof. Dr. Antonio Carlos de Oliveira Júnior

Goiânia
2020

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Amaro Galhardo, Edwardes
Implementação e Avaliação de Múltiplas Tabelas de Fluxo em Switches OpenFlow Aplicadas em vCPE e Sd-VANETs [manuscrito] / Edwardes Amaro Galhardo. - 2020.
CIII, 103 f.

Orientador: Prof. Dr. Antonio Carlos de Oliveira Júnior.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Ciência da Computação, Cidade de Goiás, 2020.
Bibliografia.
Inclui lista de figuras, lista de tabelas.

1. Multiple Flow Table. 2. SDN. 3. Sd-VANET. 4. switch OpenFlow. 5. vCPE. I. de Oliveira Júnior, Antonio Carlos, orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

ATA DE DEFESA DE DISSERTAÇÃO

Ata nº **22/2020** da sessão de Defesa de Dissertação de **Edwardes Amaro Galhardo**, que confere o título de Mestre em Ciência da Computação, na área de concentração em Ciência da Computação.

Aos vinte dias do mês de outubro de dois mil e vinte, a partir das dez horas, via sistema de webconferência da RNP, realizou-se a sessão pública de Defesa de Dissertação intitulada **"Implementação e Avaliação de Múltiplas Tabelas de Fluxo em Switches OpenFlow Aplicadas em vCPE e SD-VANETs"**. Os trabalhos foram instalados pelo Orientador, Professor Doutor Antonio Carlos de Oliveira Júnior (INF/UFG) com a participação dos demais membros da Banca Examinadora: Professor Doutor Leandro Alexandre Freitas (IFG), membro titular externo; Professor Doutor Tércio Alberto dos Santos Filho (DCC/IBiotec/UFCAT), membro titular externo. A realização da banca ocorreu per meio de videoconferência, em atendimento à recomendação de suspensão das atividades presenciais na UFG emitida pelo Comitê UFG para o Gerenciamento da Crise COVID-19, bem como à recomendação de isolamento social da Organização Mundial de Saúde e do Ministério da Saúde para enfrentamento da emergência de saúde pública decorrente do novo coronavírus. Durante a arguição os membros da banca não fizeram sugestão de alteração do título do trabalho. A Banca Examinadora reuniu-se em sessão secreta a fim de concluir o julgamento da Dissertação, tendo sido o candidato **aprovado** pelos seus membros. Proclamados os resultados pelo Professor Doutor Antonio Carlos de Oliveira Júnior, Presidente da Banca Examinadora, foram encerrados os trabalhos e, para constar, lavrou-se a presente ata que é assinada pelos Membros da Banca Examinadora, aos vinte dias do mês de outubro de dois mil e vinte.

TÍTULO SUGERIDO PELA BANCA

Documento assinado eletronicamente por **Leandro Alexandre Freitas**, **Usuário Externo**, em 20/10/2020, às 12:26, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Antonio Carlos De Oliveira Junior**, **Professor do Magistério Superior**, em 20/10/2020, às 12:26, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **EDWARDES AMARO GALHARDO**, **Discente**, em 20/10/2020, às 12:32, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Tercio Alberto Dos Santos Filho**, **Professor do Magistério Superior**, em 20/10/2020, às 13:32, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Referência: Processo nº 23070.042302/2020-89 SEI nº 1571353

EDWARDES AMARO GALHARDO

Implementação e Avaliação de Múltiplas Tabelas de Fluxo em *Switches OpenFlow* Aplicadas em vCPE e SD-VANETs

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 20 de Outubro de 2020, pela Banca Examinadora constituída pelos professores:

Prof. Dr. Antonio Carlos de Oliveira Júnior
Instituto de Informática – UFG
Presidente da Banca

Prof. Dr. Leandro Alexandre Freitas
Instituto Federal de Goiás – IFG

Prof. Dr. Tércio Alberto dos Santos Filho
Universidade Federal de Catalão – UFCAT

Graduou-se em Redes de Computadores na União Educacional do Norte - UNINORTE Acre. Durante sua graduação, trabalhou como responsável pelo departamento de informática do IAPEN, vinculada a secretaria de segurança do estado do Acre. Se especializou em Informática na Educação pela FAVENI. Atualmente é professor do Instituto Federal do Tocantins - IFTO.

Dedico esse trabalho a meu pai, Jesús Reátegui Galhardo *(in memorian)*, que através de seu exemplo em buscar o significado das coisas me induziu e incentivou a querer buscar sempre mais. E minha querida mãe, Dulcinéia Amaro Galhardo, que sempre lutou e ainda luta pelo ensino de qualidade, demonstrando isso através de seu excelente exemplo como professora e educadora que é. Dedico também a todos aqueles que me apoiaram de todas as formas possíveis, durante o período do mestrado. Todos esses me fizeram ver que quando se quer algo com qualidade, não basta apenas desejar isso, temos que correr atrás. Sem esquecer que mesmo assim, continuamos precisando uns dos outros. Por fim, quero dedicar esse trabalho ao Instituto Federal do Tocantins, IFTO, que através de sua importante política de capacitação dos servidores, tornou tudo isso possível.

Agradecimentos

Agradeço em primeiro lugar ao meu querido Deus Jeová por me dar a vida e uma boa saúde, o que considero um generoso presente de sua parte. Também me presenteou com uma valiosa família (incluindo os irmãos carnais e espirituais), que me apoiaram nos momentos mais difíceis do mestrado, que não foram poucos. Minha esposa Fernanda Galhardo, guerreira, que precisou reorganizar e adaptar sua rotina para atender as necessidades de nossas três filhas, Alícia, Sara e Mariah Galhardo, ainda menores de idade e dependentes de cuidados comuns às crianças.

Um agradecimento especial ao professor Antonio Carlos Júnior, que me orientou na condução desse trabalho, e que de maneira sábia e perspicaz conseguiu transmitir o verdadeiro sentido de uma pesquisa científica. Seus conselhos, dicas e informações, foram e serão usadas de maneira significativa nessa nova etapa de minha vida.

Agradeço também a todos os professores e professoras que fizeram parte de todo esse trabalho de transformação intelectual e emocional que o mestrado proporciona. A professora Telma Woerle, que esteve presente em quase todos as fases do curso na frente da coordenação, desde toda a condução do processo seletivo até quando foi substituída pelo professor Fábio Moreira, atual coordenador do PPGCC. Agradeço os ensinamentos e conduções iniciais dos professores Kleber Vieira e Sand Luz que puderam acompanhar e passar orientações importantes no início do curso. Agradeço ao professor Vinícius Borges pela importante contribuição que deu na consolidação do tema de minha pesquisa. Um agradecimento especial ao professor Iwens Gervásio e a professora Marcia Cappelle pelos valores transmitidos. Também agradeço a professora Sofia Anjos pelas sábias palavras, ditas em momentos oportunos. Agradeço de coração a secretaria do programa, representada pelas servidoras Mirian Castro e Mariana Rodrigues, excelentes profissionais. Não poderia deixar de mencionar a ajuda e o apoio dos colegas, alunos e companheiros de luta, Marcos Abreu, Cássio Martins, Pollyana Ribeiro, João Arce, e outros que direta ou indiretamente contribuíram de alguma forma e me ajudaram a chegar até aqui.

"Mesmo desacreditado e ignorado por todos, não posso desistir, pois para mim, vencer é nunca desistir."

Albert Einstein,
Físico Alemão.

Resumo

Galhardo, Edwardes Amaro. **Implementação e Avaliação de Múltiplas Tabelas de Fluxo em *Switches OpenFlow* Aplicadas em vCPE e SD-VANETs**. Goiânia, 2020. 100p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Com a dinamicidade e adaptabilidade proporcionada pelos paradigmas SDN e NFV, é possível fazer com que as redes de computadores se tornem programáveis. E isso faz com que o desempenho das aplicações seja otimizado, pois o gerenciamento e a resolução de possíveis problemas tornam-se fácil. Dessa forma, virtualizar funções de rede, possibilita a aplicação simultânea de diversas aplicações sobre o mesmo *hardware*. O protocolo *OpenFlow* é o grande responsável por toda essa ascensão das redes programáveis. Esse protocolo define o funcionamento de todos os elementos da rede e estabelece um padrão de interação entre eles. O plano de dados é separado do plano de controle, e o sistema operacional da rede, ou controlador, é o elemento que tem uma visão centralizada de toda a rede. Essa "visão privilegiada"permite ao controlador atualizar a arquitetura da rede sem modificar os elementos de comutação de pacotes. O *switch* é um dos elementos utilizados em uma rede para realizar o tráfego de dados através do *pipeline* de tabela de fluxo. Como as aplicações modernas necessitam cada vez mais de espaço e flexibilidade para atender a interoperabilidade exigida pelos atuais sistemas, o *pipeline* baseado em uma única tabela de fluxo pode deixar as implementações inflexíveis e rígidas. Sendo assim, este trabalho propõe a implementação de múltiplas tabelas de fluxo em *switches OpenFlow* aplicadas e avaliadas em dois cenários modernos de Redes Definidas por *Software*: vCPE (*virtual Custommer Premises Equipment*) e Sd-VANET (*Software defined Vehicular Ad-hoc Network*). Experimentos em ambientes de simulação utilizando o aplicativo *iperf* revelaram que a rede *OpenFlow* equipada com o esquema de múltiplas tabelas de fluxo, consegue manipular as funções e serviços do vCPE e da Sd-VANET com maior flexibilidade, e ainda obter um ganho médio de 17,48% no desempenho geral dos cenários propostos.

Palavras–chave

Múltipla Tabela de Fluxo, SDN, Sd-VANET, *switch OpenFlow*, vCPE.

Abstract

Galhardo, Edwardes Amaro. **Implementation and Evaluation of Multiple Flow Tables in OpenFlow Switches Applied in vCPE e SD-VANETs.** Goiânia, 2020. 100p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

With the dynamism and adaptability provided by the SDN and NFV paradigms, it is possible to make computer networks programmable. And this makes the performance of the applications to be optimized, as the management and resolution of possible problems becomes easy. In this way, virtualizing network functions, allows the simultaneous application of several applications on the same hardware. The OpenFlow protocol is largely responsible for all this rise in programmable networks. This protocol defines the functioning of all elements of the network and establishes a pattern of interaction between them. The data plan is separate from the control plan, and the network operating system, or controller, is the element that has a centralized view of the entire network. This "privileged view"allows the controller to update the network architecture without modifying packet switching elements. The switch is one of the elements used in a network to carry out data traffic through the flow table pipeline. As modern applications increasingly need space and flexibility to meet the interoperability required by current systems, the pipeline based on a single flow table can leave implementations inflexible and rigid. Therefore, this work proposes the implementation of multiple flow tables in OpenFlow switches applied and evaluated in two modern scenarios of Software Defined Networks: vCPE (virtual Custommer Premises Equipment) and Sd-VANET (Software defined Vehicular Ad-hoc Network). Experiments in simulation environments using the iperf application revealed that the OpenFlow network equipped with the scheme of multiple flow tables, can manipulate the functions and services of vCPE and Sd-VANET with greater flexibility, and still obtain an average gain of 17,48% in the general performance of the proposed scenarios.

Keywords

Multiple Flow Table, SDN, Sd-VANET, switch OpenFlow, vCPE.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

O objetivo básico de uma rede de computadores é o compartilhamento de informações, ou seja, por meio da rede, os pacotes gerados em um local de origem chegam a um determinado destino. Entretanto, o aumento exponencial dos usuários da Internet com seus respectivos casos de uso e aplicativos emergentes, fez com que as operadoras e os administradores de rede buscassem meios e recursos para se tornarem cada vez mais capacitados com o objetivo de lidar e dominar cenários e configurações cada vez mais complexos. Por exemplo, para os usuários que desejam adicionar suas próprias funcionalidades nos equipamentos de rede, existe um grande desafio: ou os equipamentos são estritamente fechados ou apenas oferecem um pequeno conjunto de opções e configurações. Consequentemente, a inovação nas redes de computadores é comprometida, obrigando as empresas a aguardar novos recursos nas atualizações de *software*, ou pior ainda, tendo que adquirir um novo equipamento de rede.

Neste contexto, as Redes Definidas por *Software* (SDN) se sobressaem, pois a *softwarização* das redes permite que se criem estruturas cada vez mais flexíveis e dinâmicas. Esse controle das redes de computadores por meio de *softwares* tornou-se mais conhecido depois da publicação do protocolo *OpenFlow* no ano de 2008. Desde então, um grande número de pesquisas e artigos estão sendo desenvolvidos pela indústria de *hardware* e pelos pesquisadores do meio acadêmico [37].

Apesar dos desafios envolvidos, personalizar os recursos e funcionalidades dos ativos de rede tais como *switches*, roteadores e outros, tem sido um dos objetivos específicos mais explorados atualmente. Dentre estes, o *switch OpenFlow* por ser um dos responsáveis pela comunicação e o transporte dos pacotes, vem sofrendo uma evolução importante e se adaptando cada vez mais ao principal objetivo das Redes Definidas por *Software*, que é o de separar o plano de controle do plano de dados. Sua tabela de fluxo, quando corretamente configurada, permite que a criação e implementação de funções de rede tais como *Firewall [34], QoS (Quality of Service) [7], DHCP (Dynamic Host Configuration Protocol) [65], NAT (Network Address Translation) [64]* e outras, se tornem um ponto forte no desempenho, e especialmente na flexibilização das ações da rede.

Portanto, mesmo proporcionando maior flexibilidade, o paradigma SDN necessita explorar outros meios de melhorar a capacidade de trabalhar de forma transparente com outros sistemas. E uma programação eficiente dos elementos de comutação tais como os *switches OpenFlow*, podem proporcionar uma melhora significativa na interoperabilidade da rede.

1.1 Motivação

Uma pesquisa realizada pela empresa Cisco Systems aponta que em 2022 haverá cerca de 50 bilhões de dispositivos conectados em todo o mundo [25]. Esse aumento exponencial de conexões, exige que os elementos responsáveis pela comunicação dos dispositivos sejam cada vez mais interoperáveis, ou seja, tenham a capacidade de trabalhar de forma transparente com outros sistemas. Além dessa capacidade, também precisam ser flexíveis no que se refere à inserção de novas funções, e sejam capazes de suportar um grande número de aplicações e suas respectivas regras. As Redes definidas por *Software* possuem essa característica. E o *switch OpenFlow*, que é um dos principais responsáveis pelo plano de dados, precisa cada vez mais trabalhar de forma transparente com outros elementos da rede. Além do mais, as informações que trafegam pelo *switch* precisam ser analisadas, processadas e encaminhadas para seus destinos de acordo com o que se demanda pelos atuais aplicativos e suas variadas funções. Entretanto, programar *switches OpenFlow* utilizando o modelo padrão de única tabela de fluxo, torna a rede limitada e muito complexa, deixando a comunicação sem a transparência necessária [52].

Em [27] e em [26], são apontadas duas categorias de casos de uso para as quais uma única tabela é muito restritiva. A primeira categoria envolve a realização de ações independentes baseadas na correspondência de diferentes campos em um pacote, o que efetivamente requer uma pesquisa separada. A segunda categoria envolve um modelo natural de processamento em duas etapas. Cada uma dessas categorias pode, em teoria, ser tratada em uma única tabela, mas o manuseio é geralmente complexo e a necessidade de combinar campos correspondentes força uma explosão de entradas de fluxo. O ponto chave é que esses casos de uso podem ser tratados de forma simples, adicionando mais tabelas de fluxo.

1.2 Contextualização do Problema

Tendo em vista o aumento explosivo na demanda por conectividade, as redes de computadores estão passando por uma grande revolução. A iminente implantação do padrão *5G* em todo o mundo faz surgir grandes desafios e oportunidades. Dentre os quais, pode-se citar a virtualização da infraestrutura da rede e a possibilidade de realizar a

programação de todo o seu comportamento. Estes avanços tornam possíveis o surgimento de projetos e a implementação de novos mecanismos capazes de gerar maior desempenho, eficiência energética e melhorar a segurança da rede e seus respectivos serviços [60].

As diversas formas de comunicação existentes e o compartilhamento de acesso aos conteúdos têm desempenhado um papel muito importante para o fornecimento de conectividade aos mais variados tipos de objetos do nosso cotidiano, o que inclusive, impulsionou o surgimento do que conhecemos como Internet das Coisas (IoT) [45]. E nesse contexto, as Redes definidas por *Software* tem o protocolo *OpenFlow* como seu principal conjunto de normas e procedimentos. Com ele é possível configurar e otimizar os fluxos de dados nas Redes definidas por *Software*.

Como nas redes baseadas no protocolo *OpenFlow* as estratégias de encaminhamento dos pacotes são determinadas por fluxo, o desenho do *switch OpenFlow* e de suas tabelas de fluxo passa a ser um dos principais problemas na construção dessas redes. Desde a especificação *OpenFlow v1.1*, o processamento de tabela de fluxo em vários estágios dentro do *switch* foi proposto para obter uma busca de tabela eficiente e flexível [43]; Entretanto, a implementação de tal proposta não foi dada, mesmo na última especificação v1.5. Cada entrada de fluxo representa um fluxo, (ou seja, pacotes com um determinado endereço *MAC*, *tag VLAN*, endereço *IP* ou porta *TCP/UDP* e assim por diante). Os pacotes de fluxo de tráfego são processados na entrada de fluxo na tabela do *switch*, e uma entrada de fluxo é identificada principalmente pelos campos de correspondência e pela suas prioridades, definidas pelo controlador *OpenFlow* [52].

No entanto, a estrutura dos campos de correspondência está se tornando cada vez mais complexa tendo em vista a nova arquitetura da Internet, os novos tipos de aplicativos e os novos formatos das mídias. Assim, a tabela de fluxo única da implementação padrão do *switch OpenFlow* pode levar ao rápido crescimento do espaço de armazenamento e, finalmente, causar o estouro da tabela [26]; Além disso, a inserção de mais e mais funções deixariam os *scripts* muito longos e tornariam sua manipulação complexa; a construção de múltiplas tabelas de fluxo podem resolver esse problema e proporcionar maior eficiência e flexibilidade.

O estado da arte está bem avançado em assuntos relacionados à virtualização de funções de rede (NFV), mais oportunamente em aplicações reais, tais como vCPE e VANETs. Com isso, surge a necessidade de atender as demandas cada vez mais crescentes de implementação de funções que possam interagir com outras funções ou dispositivos. Por esse motivo, o esquema de múltiplas tabelas no *switch OpenFlow* atende a essa necessidade. Sendo assim, a correta programação do plano de dados em SDN é de fundamental importância para o aumento da flexibilidade e o consequente aumento da interoperabilidade entre os sistemas de redes programáveis.

Com isso, podemos estabelecer que o problema motivador dessa proposta é o de

implementar e avaliar múltiplas tabelas de fluxo em cenários atuais [3]. A virtualização do equipamento de instalação dos clientes (vCPE) e as redes veiculares definidas por *software* (Sd-VANETs), tem como característica o uso de diversas funções distintas, por isso são cenários bem apropriados para se implementar os *switches* equipados com múltiplas tabelas de fluxo.

1.3 Objetivos

O objetivo principal deste trabalho é implementar e avaliar múltiplas tabelas de fluxo em *switches OpenFlow* em dois cenários de rede: vCPE e Sd-VANET.

1.4 Objetivos Específicos

- Desenvolver uma estrutura de rede no emulador Mininet-WiFi e avaliar a aplicação de múltiplas tabelas de fluxo em cenários virtuais envolvendo vCPE e Sd-VANETs.
- Caracterizar o *switch OpenFlow*, manipulando o controlador RYU para ser capaz de operar as estruturas, tanto com uma única tabela quanto com várias tabelas de fluxo.
- Gerar os casos de uso do cotidiano atual, vCPE e Sd-VANET no padrão *IEEE* 802.11, e estabelecer os parâmetros de funcionamento da rede.
- Estabelecer a conexão entre o controlador da rede e a estrutura proposta. Fazendo com que a rede deixe de ser administrada pelo controlador padrão OVS e passe a ser manipulada por um controlador externo.
- Executar os experimentos e avaliar o desempenho das simulações equipadas com as funções de rede *Firewall*, Encaminhamento, *DHCP* e *QoS*.
- Analisar e confrontar os experimentos produzidos com as estruturas equipadas com uma e com várias tabelas de fluxo, a fim de justificar o aumento da flexibilidade e da interoperabilidade da rede, quando ela for definida seguindo o modelo de múltiplas tabelas de fluxo.

1.5 Contribuições do Trabalho

Com o entendimento de que a aplicação de múltiplas tabelas no *switch OpenFlow* garante mais poder e flexibilidade à SDN, foi possível produzir e publicar o artigo intitulado *iClassMFT: Proposed Multiple Flow Tables Classes to Integrate Security and Flexibility into SDN switches*. O artigo foi publicado no *International Conference on Computational Science and Computational Intelligence (CSCI)*, em 2019 [3]. O trabalho

propõe a implementação de classes de funções de rede distribuídas em múltiplas tabelas de fluxo. Tais classes de funções proporcionariam aos desenvolvedores SDN a capacidade de manter suas estruturas mais robustas e seguras, especialmente quando se depararem com ataques do tipo *DDoS*. Esse trabalho trouxe uma contribuição importante para o correto entendimento do processo de manipulação do *switch OpenFlow* com o controlador RYU. É considerado portanto, o passo fundamental para a execução do trabalho atual.

Uma contribuição deste trabalho é a implementação do *switch OpenFlow* dotado com o esquema de múltiplas tabelas de fluxo, instanciado pelo controlador RYU [26]. Com esse arranjo é possível executar a avaliação e experimentação de um ambiente simulado dos equipamentos de instalação dos clientes (vCPE). Essa avaliação é importante, pois revela a necessidade de se criar estruturas robustas e flexíveis, capazes de atender as atuais demandas da indústria e dos próprios clientes. O que inclui a capacidade da rede em conseguir lidar com os mais variados tipos de dados provenientes dos diversos tipos de mídias existentes na atualidade.

Outra importante contribuição deste trabalho é a de simular uma Sd-VANET dentro do padrão de redes sem fio para ambientes veiculares WAVE (*Wireless Access in Vehicular Environments*) ou 802.11p, e fazer com que a comunicação, bem como a inserção de funções nessa rede seja conduzida por um controlador centralizado capaz de fornecer serviços de forma dinâmica e flexível. Essa abordagem é fundamental nos casos em que se pretende simular um ambiente veicular que busque oferecer uma maior disponibilidade de um veículo de segurança ou socorro, por exemplo, em situações onde o tempo do atendimento de uma emergência é, na maioria dos casos fundamental, para se garantir o sucesso da ocorrência. Assim, a abordagem de criação de estruturas operando com *switches* equipados com múltiplas tabelas, possibilitam a inserção de mais funções de rede em ambientes cuja demanda é cada vez maior.

1.6 Estrutura do Texto

Este trabalho está organizado da seguinte forma: Os conceitos sobre SDN, NFV, protocolo *OpenFlow, switches OpenFlow*, Tabelas de Fluxo, virtualização dos equipamentos de instalação dos clientes (vCPE) e as Sd-VANETs, bem como suas principais características e os fatores que envolvem a comunicação da estrutura através de redes *WiFi* padrão 802.11, são apresentados no Capítulo 2. Os trabalhos relacionados mais relevantes são apresentados no Capítulo 3. O Capítulo 4 apresenta a proposta, a metodologia do desenvolvimento da pesquisa e os resultados obtidos após a realização dos experimentos. A discussão referente aos resultados obtidos são apresentados no Capítulo 5. E finalmente, no Capítulo 6, são inseridas as considerações finais deste trabalho e as sugestões de trabalhos futuros.

CAPÍTULO 2

Fundamentos Teóricos

Ao analisar o que existe na literatura referente a programação de redes de computadores, especificamente a programação do *switch OpenFlow*, constata-se a necessidade de entender os conceitos norteadores que estabelecem os novos paradigmas de rede, juntamente com seus equipamentos e aplicativos. Para tanto, apresentamos os conceitos de SDN, NFV, do protocolo *OpenFlow*, do vCPE e da Sd-VANET.

2.1 Redes definidas por *Software* (SDN)

Nas redes tradicionais, para realizar uma configuração específica, os administradores precisam configurar de forma individual cada dispositivo da rede, e na maioria das vezes precisam utilizar comandos de baixo nível, específicos, de acordo com seu respectivo fornecedor, o que acaba deixando a rede muito complexa e difícil de gerenciar. Além disso, a rede precisa ser capaz de suportar possíveis falhas e conseguir se adaptar as eventuais mudanças de carga em seus enlaces físicos ou lógicos [56].

Em SDN, a ideia principal é separar o controle lógico da rede (plano de controle) dos *switches* e roteadores que encaminham o tráfego (plano de dados). Neste caso os *switches* se tornam dispositivos simples de encaminhamento e a lógica de controle é implementada em um controlador centralizado, o que torna mais simples o processo de avaliação e reconfiguração da rede [36].

De acordo com [36] a centralização lógica da rede fornecida pelo paradigma SDN, constitui ao menos três importantes vantagens: primeiro, torna o processo de configuração da rede mais simples e menos propenso a erros, o que é justificado pelo fato de as políticas de operação da rede serem realizadas em linguagens de alto nível e componentes de *software*; Em segundo lugar, um programa de controle pode reagir automaticamente a mudanças espúrias do estado da rede e, assim, manter intactas as políticas de alto nível. Em terceiro lugar, a centralização da lógica de controle em um controlador com conhecimento global do estado da rede simplifica o desenvolvimento de funções, serviços e aplicativos de rede mais sofisticados. A Figura 2.1 apresenta uma ideia geral do paradigma SDN.

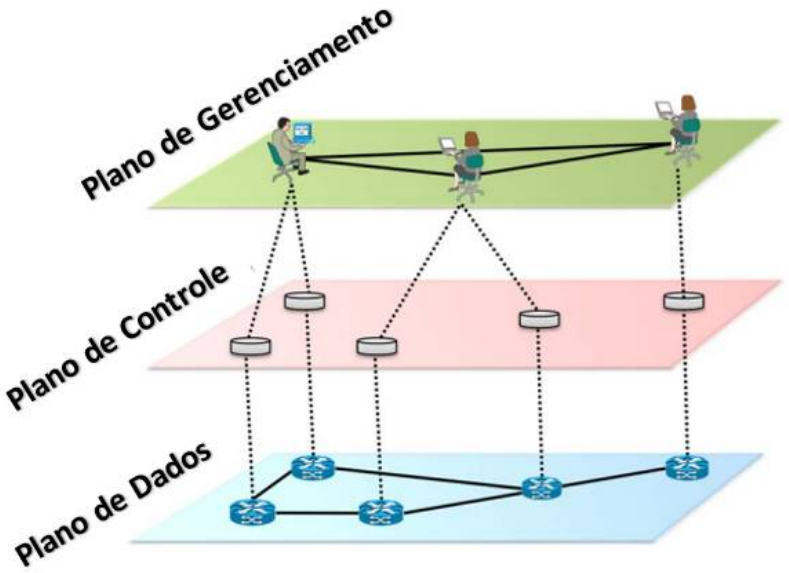

Figura 2.1: *Separação dos planos de Gerenciamento, Controle e Dados. Uma visão em camadas da funcionalidade da rede. Adaptado de* [36]

Conforme apresentado na Figura 2.1, as redes de computadores podem ser divididas em três planos de funcionalidade: os planos de dados, controle e gerenciamento. O plano de dados corresponde aos dispositivos de rede, que são responsáveis por encaminhar (eficientemente) os dados. O plano de controle representa os protocolos usados para preencher as tabelas de encaminhamento dos elementos do plano de dados. O plano de gerenciamento inclui os serviços de *software*, tais como ferramentas baseadas em protocolo de gerenciamento de rede simples (SNMP), usadas para monitorar e configurar remotamente a funcionalidade de controle [50]. A política de rede é definida no plano de gerenciamento, o plano de controle impõe a política e o plano de dados a executa, encaminhando os dados corretamente. Em redes *IP* (*Internet Protocol*) tradicionais, os planos de controle e de dados são fortemente acoplados, incorporados nos mesmos dispositivos de rede e todo a estrutura era altamente descentralizada. Isso foi considerado importante para o *design* da Internet nos primeiros momentos, pois parecia a melhor maneira de garantir a resiliência da rede, que era um objetivo crucial do design [36].

2.2 Protocolo *OpenFlow*

Em SDN uma rede inteira pode ser configurada de forma dinâmica sem a necessidade de configurar cada um dos equipamentos. Isso ocorre devido a aplicações programadas em *software* através do controlador centralizado da rede. Este controlador será mais bem detalhado posteriormente na Seção 2.2.1. Entretanto, para se estabelecer

a comunicação do plano de controle com o plano de dados, foi necessário criar, e em seguida padronizar uma Interface de Programação de Aplicativos - API (*Application Programming Interface*), sendo estabelecido então o protocolo *OpenFlow* [44].

O protocolo *OpenFlow* foi introduzido comercialmente em 2008 pela empresa Nicira, e em 2012 a empresa foi comprada pela *VMware*. O objetivo inicial do protocolo é proporcionar o isolamento do tráfego de produção do tráfego experimental. Isso faz com que os pesquisadores da área de redes executem seus experimentos sem interromper redes que estejam em produção. O protocolo permite a comunicação entre o controlador e os dispositivos que encaminham os pacotes, definidos como *switches OpenFlow* [44] [52]. A Figura 2.2 apresenta a arquitetura SDN utilizando o protocolo *OpenFlow*.

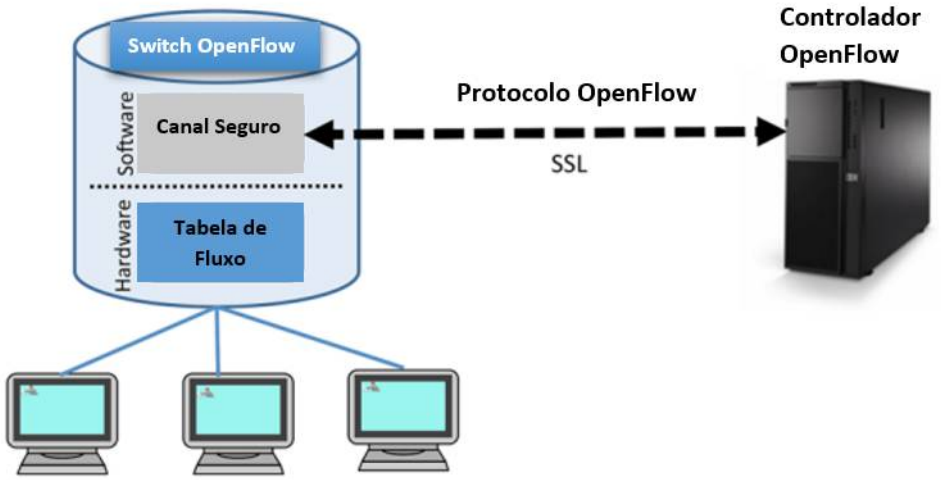

Figura 2.2: *Ideia geral do paradigma SDN. Adaptado de* [52]

A Figura 2.2 mostra um *switch OpenFlow* e um controlador *OpenFlow*. O termo *"switch"* é usado para os nós do *OpenFlow* porque, como veremos, os caminhos são determinados pelo controlador.

O *switch OpenFlow* tem duas partes: a parte que contém as filas, os transmissores de quadro e receptores de quadro, com as tabelas de fluxo associadas a eles, e a segunda parte que governa a comunicação com o controlador usando o protocolo de sinalização *OpenFlow*.

A primeira parte contém todos os elementos necessários para o transporte físico dos *frames* do nó e também contém as tabelas usadas para direcionar os fluxos para a fila de saída correta. Pode haver uma tabela para cada fluxo, como uma tabela para um conjunto de fluxos multiplexados no mesmo caminho. O *OpenFlow* fornece os elementos necessários do controlador para criar, gerenciar e destruir as linhas da tabela de fluxo, que também pode ser chamada de tabela de comutação. A segunda parte diz respeito à

comunicação entre o nó e o controlador, e as informações que devem ser enviadas entre eles [52].

O *OpenFlow* usa um canal SSL/TLS (*Secure Socket Layer / Transport Layer Security*) seguro para autenticar ambas as extremidades da comunicação, o que reduz muito o risco de ataque à comunicação em andamento e exige autenticação mútua. *OpenFlow* oferece os meios de identificar os fluxos de pacotes usando informações de nível 1, 2, 3 e 4. O protocolo também transporta ações para indicar o que precisa ser feito para o fluxo. Por fim, ele fornece estatísticas muito precisas ao controlador para que o algoritmo de determinação do caminho possa fazer seu trabalho com um conhecimento quase perfeito do estado da rede [52].

A Figura 2.2 apresenta portanto a arquitetura de um *switch OpenFlow* estabelecendo comunicação com um controlador *OpenFlow*. Apresentamos a seguir as informações relativas aos três elementos da arquitetura: Controlador, Canal Seguro e Tabela de Fluxo.

2.2.1 Controladores *OpenFlow*

Fornecendo uma visão centralizada, o controlador é o elemento que gerencia e controla toda a SDN. É responsável por tomar as decisões da rede, como por exemplo, adicionar ou remover as entradas dos pacotes nas tabelas de fluxo instanciadas nos *switches*. De maneira geral os controladores podem fazer com que os *switches* executem as seguintes ações:

• Reescreve o endereço dos pacotes recebidos ou transfere os pacotes para uma porta diferente da especificada;

• Transfere os pacotes recebidos para o controlador (*Packet-In*);

• Transfere os pacotes encaminhados pelo controlador da porta especificada (*Packet-Out*) [52].

É possível ainda fazer com que o *switch* execute as três ações listadas anteriormente apenas por executá-las de forma combinada. Em primeiro lugar, é preciso usar a função *Packet-In* para aprender os endereços *MAC* (*Media Access Control*). Então o controlador pode usar a função *Packet-In* para receber os pacotes do *switch*. Em seguida, o *switch* analisa os pacotes recebidos para aprender o endereço *MAC* do *host* e as informações sobre a porta conectada. Após o aprendizado, o *switch* transfere os pacotes recebidos e verifica se o endereço *MAC* de destino dos pacotes pertence ao *host* aprendido. Dependendo dos resultados dessa análise, é que o *switch* realiza o processamento. Duas ações poderão então ocorrer:

• Se o *host* já foi aprendido... Usa a função *Packet-Out* para transferir os pacotes para a porta de destino.

• Se o *host* for desconhecido... Usa-se a função *Packet-Out* para realizar *flood* e enviar para todas as portas. [54].

Atualmente existem vários controladores *OpenFlow* [57], dentre os quais pode-se citar o NOX [19], POX [24], ONOS [28], FloodLigth [23] e RYU [54]. Baseado na linguagem de programação C/C++, o NOX foi o primeiro controlador *OpenFlow* a disponibilizar uma API capaz de oferecer uma visão da topologia de rede dos *switches* e demais enlaces. Seu sucessor, o POX, é um controlador *OpenFlow* escrito em *Python* com uma interface SDN de alto nível. O controlador ONOS (*Open Network Operation System*) também é um controlador de código aberto que tem por objetivo fornecer uma arquitetura tolerante a falhas. Desenvolvido pela equipe do ON.Lab, que é apoiada por grandes nomes do mercado, tais como AT&T, Cisco, Fujitsu, Intel e outros. Já o controlador FloodLight foi criado com base no código fonte do controlador Beacon. Sua utilização está mais voltada para os ambientes comerciais. O RYU é um controlador de código aberto, escrito em *Python*, e é capaz de fornecer componentes de software com interfaces de programação de aplicação bem definida, o que favorece a criação, gerenciamento e controle de toda a rede.

Por questões de conveniência e pela boa interação com o emulador Mininet-WiFi, este trabalho utiliza o controlador RYU para realizar os experimentos [15]. A Figura 2.3 apresenta o modelo de programação de aplicações do RYU.

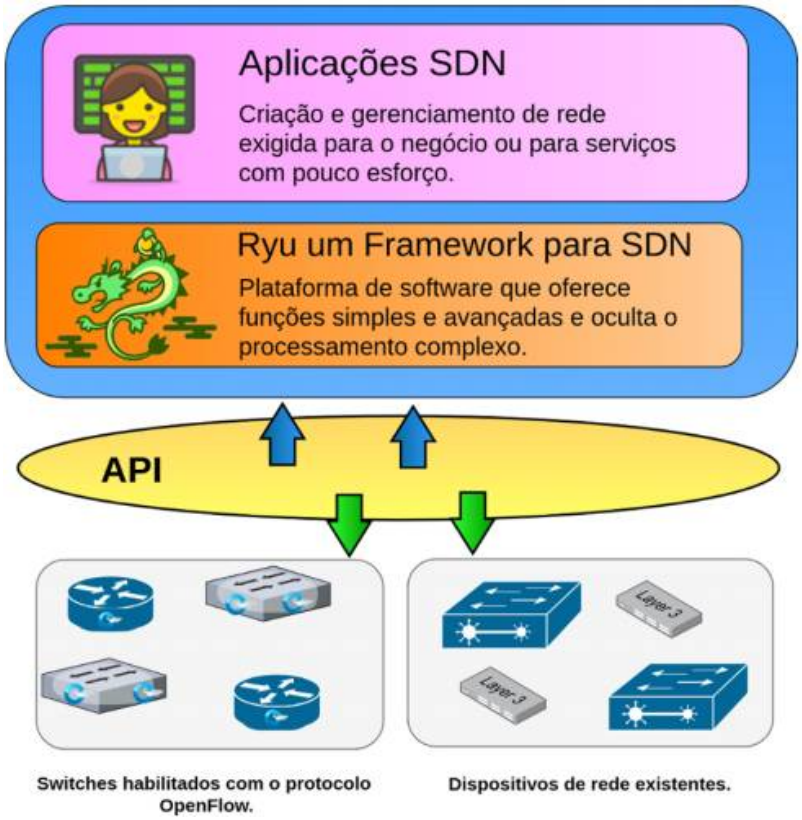

Figura 2.3: *Uma visão da arquitetura do controlador RYU*. Adaptado de [59]

A Figura 2.3 apresenta uma visão da arquitetura do controlador RYU, onde a comunicação entre as camadas ocorre através de uma API. Em primeiro lugar se apresenta a camada responsável pela criação e gerenciamento da rede, denominada Aplicações SDN. Em seguida, na camada de controle, está o controlador RYU, oferecendo funções simples e avançadas, ocultando o processamento complexo. E por último, a camada que pode conter tanto os *switches* quanto os demais dispositivos de rede habilitados para o protocolo *OpenFlow* [59].

2.2.2 Canal Seguro

O estabelecimento de um canal seguro com o controlador é uma parte essencial para que haja a caracterização de um *switch OpenFlow*. É por meio desse canal que o controlador irá gerenciar o *switch*. Para que haja uma autenticação e se garanta a confidencialidade da comunicação em ambas as extremidades, o *OpenFlow* usa um canal seguro SSL/TLS que cria um canal criptografado entre um servidor e um terminal, garantindo que todos os dados transmitidos sejam sigilosos e seguros. Isso reduz consideravelmente o risco de ataque à comunicação que pode estar em andamento.

De acordo com [44], quando se estabelece uma comunicação *OpenFlow*, tanto o controlador quanto o *switch* devem enviar imediatamente uma mensagem do tipo *Hello* (*OFPT-HELLO*), contendo a mais alta versão do protocolo *OpenFlow* suportada pelo dispositivo. Quando receber a mensagem, o dispositivo deve escolher a menor versão do *OpenFlow* entre a que foi enviada e recebida. Se houver compatibilidade entre as versões, a comunicação continua normalmente. Se não houver, é gerada uma mensagem de erro do tipo *OFPT-ERROR* e a comunicação é encerrada de imediato.

Ainda de acordo com [44], caso haja alguma perda de conexão entre o *switch* e o controlador, o *switch* tentará se conectar a um controlador *back-up*, caso exista. Se essa conexão também falhar, o *switch* entrará em modo de emergência, modo esse que utilizará apenas as entradas na tabela de fluxos marcadas com um bit de emergência e todas as outras entradas serão deletadas. Essa situação ainda ocorre nas implementações atuais, o que está confirmado na literatura [52].

2.2.3 Tabelas de Fluxo

O protocolo *OpenFlow* utiliza as tabelas de fluxo para realizar o encaminhamento dos pacotes com base nas entradas presentes na tabela. Além disso, a tabela de fluxo é composta de regras, ações e contadores estatísticos. Nos *switches OpenFlow* todos os pacotes são processados pelo *pipeline OpenFlow* e não podem ser processados de outra forma. A Figura 2.4 apresenta as entradas de tabela presentes na versão mais atual do *OpenFlow*, a versão 1.5 [27].

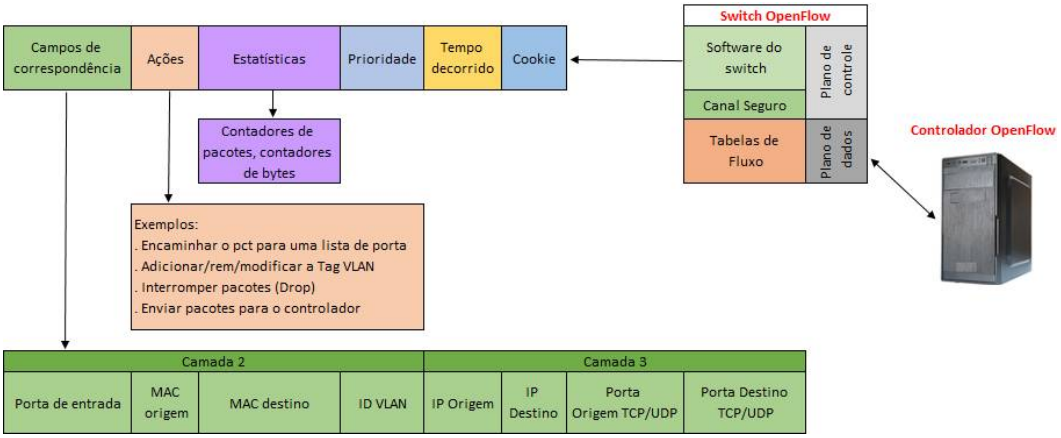

Figura 2.4: *Principais Componentes de uma Entrada de Fluxo em uma Tabela de Fluxo OpenFlow. Extraído de* [27]

De acordo com a Figura 2.4, cada entrada de tabela de fluxo contém:

• Campos de correspondência: Realizam as possíveis combinações (*maching*) com os pacotes. Estes consistem na Porta de Entrada e Cabeçalhos de pacote, e, opci-

onalmente, outros campos do *pipeline*, como metadados especificados por uma tabela anterior;

- Prioridade: Se refere a precedência correspondente à entrada de fluxo;
- Estatísticas, que são atualizados quando os pacotes são correspondidos;
- Ações: Utilizadas para modificar o conjunto de ação ou o processamento do *pipeline*;
- Tempo decorrido: Especifica a quantidade máxima de tempo ou o tempo ocioso antes do fluxo expirar no *switch*;
- *Cookie*: É o valor de dados escolhido pelo controlador que pode ser usado pelo para filtrar entradas de fluxo afetadas pelas estatísticas de fluxo, modificação de fluxo e solicitações de exclusão de fluxo. Não usado ao processar pacotes;

Uma entrada de tabela de fluxo é identificada por seus campos de correspondência (*match*) e de prioridade, isso significa que os campos onde ocorrem as correspondências juntamente com as prioridades, identificam uma única entrada de fluxo em uma tabela de fluxo específica. A entrada de fluxo que *curinga* (*) todos os campos, ou seja, todos os campos omitidos, e tem prioridade igual a zero é chamada de entrada de fluxo de tabela vazia (*table-miss*) [27].

Uma instrução de entrada de fluxo pode conter ações a serem executadas no pacote em algum ponto do *pipeline*. Essas ações são extraídas de cada pacote, que normalmente incluem vários campos de cabeçalho de pacote, como endereço de origem e destino *Ethernet*, endereço de origem e destino *IPv4*, endereço *MAC* de origem e destino, dentre outros. Em seguida o *switch* começa realizando uma busca de tabela na primeira tabela de fluxo e, com base no processamento do *pipeline*, pode realizar buscas de tabela em outras tabelas de fluxo [27]. A Figura 2.5 apresenta o fluxo do pacote dentro do *pipeline* do *switch*.

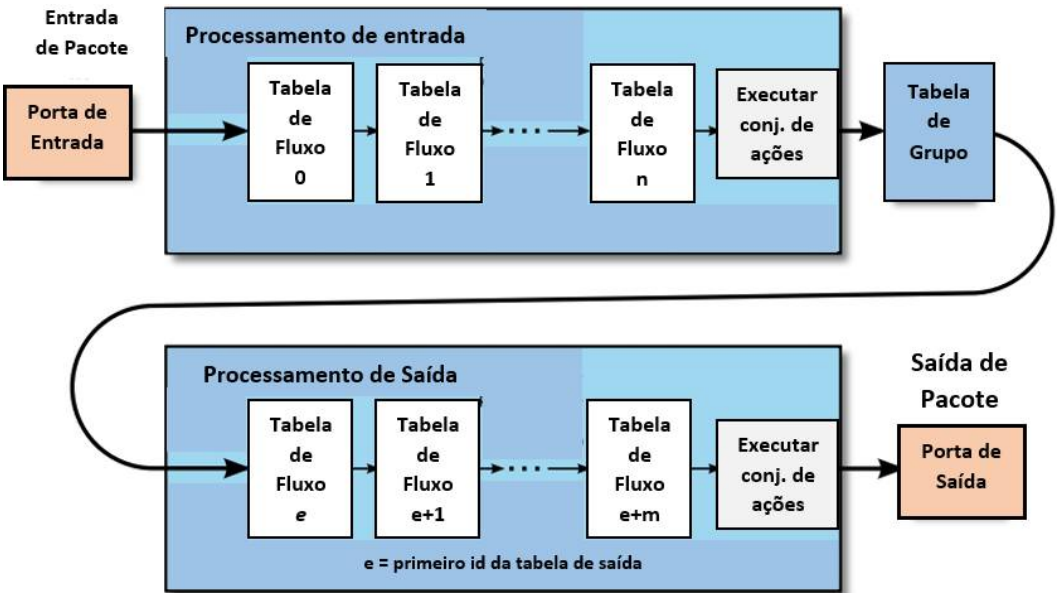

Figura 2.5: *Fluxo de pacotes através do pipeline de processamento. Adaptado de* [27]

2.3 Virtualização das Funções de Rede (NFV)

O SDN torna possível o desenvolvimento de novas soluções de rede. Nessa perspectiva, surgiu um novo paradigma complementar ao SDN, que é a tecnologia conhecida por Virtualizar as Funções de Rede (*Network Function Virtualization* - NFV) [67]. As funções de rede tradicionais são virtualizadas e executadas em equipamentos genéricos por meio deste novo paradigma.

Um dos principais objetivos da tecnologia NFV é a redução dos custos operacionais e de mão de obra, conhecidos como CAPEX (*CAPital EXpenses*) e OPEX (*OPerational EXpenses*) [14]. Outro benefício é a eliminação de dispositivos físicos dedicados, o que permite a adoção de novos serviços e funções, além da proporcionar maior elasticidade, escalabilidade e flexibilidade. Por isso que, diante de tantos benefícios, especialistas e pesquisadores da indústria e da academia estão cada vez mais interessados em buscar métodos e novas soluções para planejar, implantar e fazer com que as redes de computadores se mantenham operacionais [42].

Os equipamentos de rede físicos, também conhecidos como *middleboxes*, estão sendo substituídos por seus correspondentes virtualizados implementados em *software*, sem a necessidade de um *hardware* especializado para executar suas funções virtuais [20]. Sua implementação é feita em *hypervisors*, que são *softwares* que criam máquinas virtuais (*Virtual Machine* - VM) capazes de compartilhar seus recursos virtualizados [6]. Existem basicamente, dois tipos de *hypervisors*, os do tipo *Bare Metal*, que se executa diretamente sobre o *hardware* e os *hosted*, que se executa sobre um sistema operacional.

O *Oracle VirtualBox* e o *VMware Player* são uns dos exemplos mais conhecidos do tipo de *hypervisors* [67].

O ETSI (*European Telecommunications Standards Institute*) define uma estrutura arquitetônica NFV que permite que as funções de rede virtualizadas (*Virtualized Network Functions* - VNFs) sejam implantadas e executadas em uma infraestrutura NFV na camada NFVI (*Network Function Virtualization Infrastructure*), que consiste em recursos de *hardware* COTS (*Commercial Off-The-Shelf*), incluindo computação, armazenamento e rede, embrulhado com uma camada de *software* que abstrai e particiona logicamente. Em implementações baseadas em *hypervisor*, uma VNF é tipicamente mapeada para uma máquina virtual (*VM*) na camada NFVI, mas ela também pode ser dividida em vários Componentes de Funções de Rede Virtualizadas (*Virtualized Network Functions Components* - VNFCs) carregados em *VMs* separados (por exemplo, com diferentes requisitos de escala). A implantação, execução e operação de VNFs em um NFVI são orientadas por um sistema de gerenciamento e orquestração (*systems management and orchestration* - SMO), o comportamento é impulsionado por um conjunto de descritores de metadados (também conhecidos como descritores de NFV) descrevendo as características dos serviços de rede e seus VNFs constituintes. O sistema de SMO inclui um orquestrador NFV (*NFV-Orchestrator* - NFVO) responsável pelo ciclo de vida dos serviços de rede, um conjunto de gerentes de VNF encarregados do ciclo de vida dos VNFs (incluindo o dimensionamento/expansão do VNF) e um gerenciador de infraestrutura virtualizado (*Virtualized Infrastructure Manager* - VIM), que pode ser visto como um sistema de gerenciamento de nuvem estendido responsável pela alocação e liberação de recursos NFVI mediante solicitação do VNFM (*Network Functions Virtualization Manager*) e NFVO [42].

2.3.1 vCPE

A sigla CPE (*Customer Premises Equipment*) é um termo técnico bem difundido e bastante utilizado por operadoras de telecomunicações e fornecedores de serviços de comunicação [51]. Em outras palavras o termo se refere ao equipamento dentro das instalações do cliente. Para uma operadora de serviços celular o CPE é o telefone celular, para uma empresa de telefonia, o CPE pode ser o aparelho de telefone, para serviços de voz, ou o *modem* ADSL (*Assymetrical Digital Subscriber Line*), para serviços de dados. Para um cliente comum, CPEs podem ser *switches*, roteadores, *modems*, receptores, antenas, etc. Ou seja, qualquer equipamento que seja necessário para um cliente receber o serviço de comunicação, é um CPE.

Virtualizar esses equipamentos (*Virtual Customer Premises Equipment* - vCPE) envolve a virtualização do equipamento dentro das instalações do Cliente (CPE) [46]. A

virtualização depende do *software* para simular a funcionalidade do *hardware* e criar um sistema de computador virtual. E existem várias vantagens em se fazer isso. Uma delas é que as organizações de TI (Tecnologia da Informação) podem executar mais de um sistema virtual, podem, por exemplo, ter vários sistemas operacionais e aplicativos em um único servidor. E com isso, os servidores passam a ter um maior rendimento, redução dos custos com configuração, refrigeração, manutenção, e a redução da necessidade de aquisição de inúmeros equipamentos.

As máquinas virtuais devem ser posicionadas no *Data Center*, localizado na rede local, ou no PoP (Ponto de Presença) do operador que gerencia a interface de acesso à rede local, ou em um dos *Data Centers* da operadora, ou ainda, em um dos *Data Centers* da empresa usando a rede da operadora para acessá-la [51]. Podemos ver um exemplo do vCPE na Figura 2.6 em que as principais máquinas virtuais estão posicionadas de forma híbrida na LAN de uma rede principal.

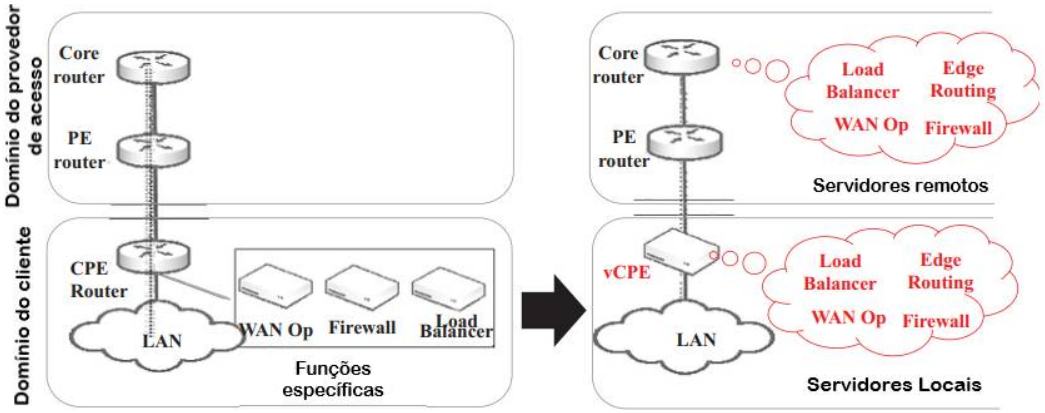

Figura 2.6: *Exemplo de um vCPE com uma arquitetura híbrida. Adaptado de* [51]

É importante destacar que os vCPEs são controlados pelas Sd-WANs (*Software-defined Wide Area Network*), que é uma solução responsável por gerenciar várias WANs (Wide Area Networks) de uma determinada empresa. Sd-WAN ou Redes *Wireless* definidas por *Software*) é uma solução para controlar e gerenciar vários WANs de uma empresa utilizando a tecnologia SDN. Como em SDN, o plano de dados e o plano de controle estão separados, um controlador centralizado deve ser adicionado para gerenciar fluxos, roteamento ou políticas de *switch*, prioridade de pacotes, políticas de rede, etc. A tecnologia SD-WAN é baseada na sobreposição, o que significa nódulos que representam redes subjacentes [51].

Cada vez mais, esses dois produtos, SD-WAN e vCPE, são integrados em um único produto. As máquinas virtuais que realizam o CPE são gerenciadas pelo controlador SD-WAN. Dessa forma, um serviço pode ser instanciado e levar em conta tanto a

participação da rede local quanto a compartilhamento das redes WAN. Isso nos permite gerenciar a qualidade do serviço e a segurança de ponta a ponta [46].

2.3.2 VANETs

As Redes Veiculares *Ad-hoc* (*Vehicular Ad-hoc NETwork* - VANET) surgiram com o objetivo de se estabelecer uma comunicação eficiente entre os veículos devido a necessidade de soluções inteligentes capazes de lidar com o expressivo aumento do número de veículos presentes nas ruas, cidades e rodovias, o que acaba produzindo grandes complicações rodoviárias [8]. As VANETs foram criadas baseadas no principio das Redes *Ad-hoc* móveis - MANETS (*Mobile Ad-hoc Network*), que são redes sem fio criadas de forma espontânea para a troca de dados entre nós móveis que se comunicam independente da existência de uma infraestrutura, bastando para isso que o nó destino esteja ao alcance do emissor. A Figura 2.7 apresenta a definição de uma MANET [66].

Figura 2.7: *Ilustração da ideia geral de uma MANET. Adaptado de* [8]

De acordo com a Figura 2.7, os nós (*notebooks, smartphones* e outros), todos os terminais funcionam como roteadores, encaminhando de forma comunitária as comunicações advindas dos terminais vizinhos. Um dos protocolos usados para essas redes *ad hoc* sem fio é o OLSR (*Optimized Link State Routing Protocol*) [9].

Ad hoc é uma expressão latina que significa "para esta finalidade"ou "com este objetivo". Geralmente se refere a uma solução destinada a atender a uma necessidade específica ou resolver um problema imediato - e apenas para este propósito, não sendo aplicável a outros casos. Portanto, tem um caráter temporário. Em um processo *ad hoc*, nenhuma técnica de uso geral é empregada pois as fases variam a cada aplicação, conforme a situação assim o requeira. O processo nunca é planejado ou preparado antecipadamente.

Aplicando o princípio das MANETs, as VANETs são um tipo moderno de rede sem fio que permitem a comunicação entre os veículos (*Intervehicle Communication*). E essa comunicação pode ser realizada de várias maneiras diferentes, por exemplo: Veículo-para-veículo (*Vehicle-to-Vehicle* - V2V), veículo-para-pedestre (*Pedestrian Vehicle* - V2P), veículo-para-infraestrutura (*Vehicle-to-Infrastructure* - V2I) ou o mais recente, veículo-para-qualquer-coisa (*Vehicle-to-Everything* - V2X) [63] [16] [61]. A conectividade em VANET usa o padrão *IEEE* 802.11 na frequência de 5,9 GHz. Em termos de serviço de infoentretenimento, a VANET deverá oferecer suporte a informações como transferência de dados multimídia e acesso à Internet [18].

Então, basicamente esta tecnologia usa carros como nós móveis para criar uma rede móvel e transformar todos os carros participantes da rede em um roteador ou nó sem fio, recebendo e enviando pacotes uns aos outros. Com isso, carros que estão a uma distância aproximada de 100 a 300 metros um do outro podem se conectar e formar uma rede de amplo alcance [21]. Nós móveis são os sensores incorporados nos veículos que são chamados de unidades a bordo (*Onboard Unit* - OBU). E as unidades usadas para processamento de sinal e compartilhamento de dados de forma fixa são chamadas de Unidades a beira da estrada (*RoadSide Unit* - RSU).

Além das VANETs poderem ser utilizadas no sistema de transporte inteligente (*Intelligent Transportation System* - ITS), também podem ser usadas na disseminação simples de informação, como as mensagens de conscientização cooperativa (*Cooperative Awareness Messages* - CAMs), e ainda nas disseminações de mensagens de múltiplos canais em longas distâncias [32]. Dessa forma são capazes de fornecer uma forte ajuda tecnológica para automóveis que usam o *Wi-Fi* para transmitir informações gerais de segurança, especialmente serviços de segurança viária que tem o importante objetivo de reduzir o número de acidentes rodoviários através do compartilhamento de dados pela internet [18].

A Figura 2.8 apresenta a representação básica de uma VANET onde os veículos estabelecem comunicação ente si (*V2V*) e entre as infraestruturas (*V2I*) [47] [63].

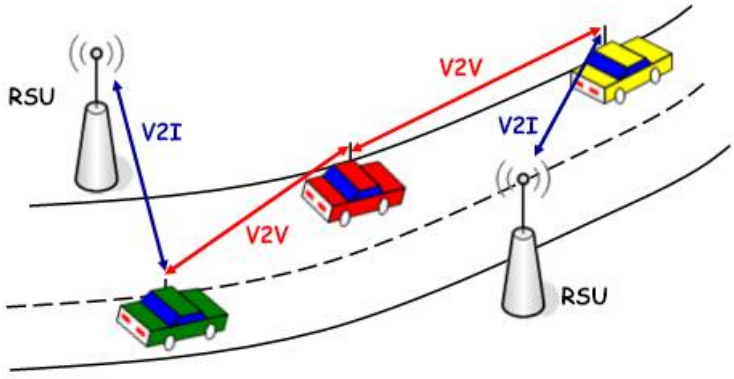

Figura 2.8: *Ilustração da ideia geral de uma VANET. Extraído de* [47]

2.3.3 SD-VANETs

Atualmente, o aumento do número de veículos tem causado problemas. Um deles é um engarrafamento que muitas vezes ocorreu devido a acidentes, então esses problemas levam à necessidade de um sistema tecnológico que possa nos ajudar a reduzir esses efeitos negativos. O Sistema de Transporte Inteligente (*Intelligent Transportation System* - ITS), que se apresenta como uma das respostas promissoras, é uma combinação de sistema de transporte inteligente com tecnologia da informação para melhorar a acessibilidade, eficiência e segurança do transporte. A tecnologia ITS poderia fornecer informações em tempo real aos usuários das estradas relacionadas à situação da estrada, como quando há acidentes de trânsito ou congestionamentos ocorridos em uma determinada área viária. A presença dessa tecnologia poderia dar soluções ou alternativas para os usuários das estradas, e pode evitar os engarrafamentos. O sistema ITS também pode fornecer informações sobre a condição dos veículos existentes circulando nas proximidades, para que possa ajudar os usuários a evitar o acidente. Uma das tecnologias do sistema ITS que ainda está em desenvolvimento é a *Vehicular Ad-Hoc Network* (VANET) [41].

Os dispositivos de rede têm vários controles e operações de fluxo de dados no mesmo dispositivo. Um dos controles é o plano de gerenciamento de rede. Este plano é usado para configurar cada nó da rede separadamente. A natureza estática da rede atual não permite plano de configuração de controle total. O conceito principal do SDN é oferecer gerenciamento de controle para que os usuários gerenciem o encaminhamento de *hardware* de cada elemento da rede. A coisa mais fácil que SDN poderia prover para VANET é tornar suas unidades na beira da estrada (*RoadSide Unit - RSU*) habilitado para SDN, por exemplo, usando um controlador como em *switch OpenFlow*. Além disso, o escopo do controlador pode ser estendido às unidades de bordo (*OnBoard Unit* - OBU) que podem atuar como usuários finais e podem ser abstraídas como elementos incluídos

nos dados, como ocorre nas RSUs e outros nós da infraestrutura. Portanto, a OBU pode ser acionada pelo controlador para seu desempenho, como a implantação de dados *V2V* de vários saltos [13].

2.4 Conclusão

Apresentamos neste capítulo os principais fundamentos envolvidos com o tema desta dissertação. Descrevemos uma visão geral dos principais conceitos e fundamentos sobre SDN e a construção do esquema de múltiplas tabelas de fluxo no *switch* baseado no protocolo *OpenFlow*. Discutimos as principais abordagens, envolvendo virtualização das funções de rede, o que inclui o vCPE e Sd-VANETs. Essa visão geral dos principais conceitos envolvidos em SDN e no protocolo *OpenFlow* servem como *background* para a formulação do problema motivador de nossa proposta, que é o de implementar e avaliar múltiplas tabelas de fluxo em cenários envolvendo vCPE e Sd-VANETs. No próximo capítulo, apresentamos os trabalhos relacionados relevantes para o desenvolvimento desta pesquisa.

CAPÍTULO 3

Trabalhos Relacionados

Nesta seção são apresentados alguns trabalhos relacionados, que são relevantes para a execução desta dissertação.

3.1 Experimentos utilizando Múltiplas Tabelas de Fluxo (MTF)

Com o propósito de impedir o rápido crescimento do espaço de armazenamento e consequentemente o estouro da tabela de fluxo, CHEN, Z. *et al* [11] propuseram a implementação de várias sub-tabelas de fluxo dentro de uma única tabela principal.

A Figura 3.1 apresenta a metodologia proposta, onde a tabela de fluxo original é subdividida em vários outros sub-fluxos, o que, de acordo com os autores, otimiza o espaço de armazenamento da tabela no *switch*, gerando com isso um importante ganho de armazenamento, em especial nos equipamentos que possuem memória *TCAM* (*Ternary Content Access Memory*). As referidas vantagens foram constatadas após se realizar testes comparativos entre os equipamentos configurados com uma tabela de fluxo padrão e a tabela de fluxo otimizada com o modelo de sub-fluxos.

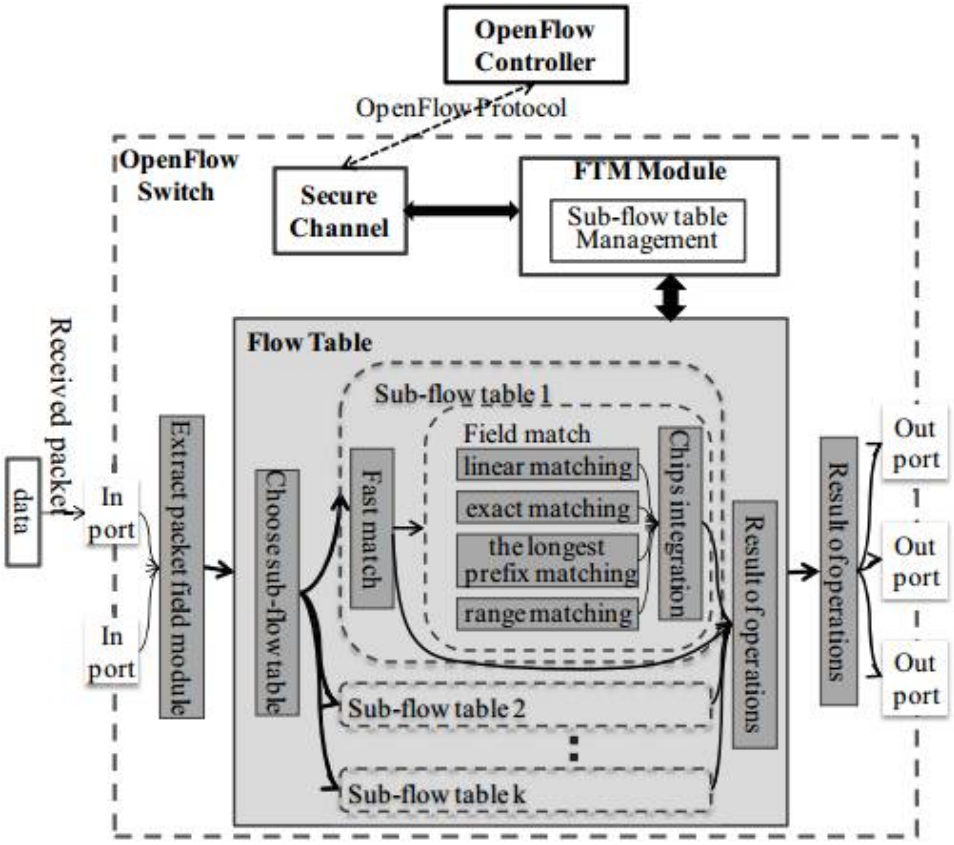

Figura 3.1: *Tabela de Fluxo OpenFlow otimizada pelo algoritmo H-SOFT e dividida em sub-fluxos. Extraído de* [11]

Conforme apresentado na Figura 3.1, no modelo de pesquisa (*lookup*) proposto, o módulo de gerenciamento de tabelas de fluxo (FTM) é adotado para gerenciar as tabelas de fluxo. Este módulo é independente em cada *switch* e é transparente para o controlador *OpenFlow*. O controlador pode emitir regras com o mesmo formato para diferentes *switches*. O módulo FTM pode adotar diferentes algoritmos de gerenciamento como o H-SOFT [17]. Este módulo é escalável e pode definir funções obrigatórias e opcionais, onde neste trabalho é necessária a gestão da tabela de subfluxo

De maneira similar GE, J. *et al* [17] propuseram a redução do espaço de armazenamento da tabela de fluxo, e também apresentaram algoritmo *H-SOFT* para converter uma única tabela de fluxo com campos de correspondência complexos e de alta dimensão em múltiplas tabelas de fluxo com campos de correspondência simples e de baixa dimensão. Entretanto, os experimentos que levaram aos resultados foram realizados em equipamento físicos, o que limita a quantidade de aplicações que podem se beneficiar da redução do tamanho das tabelas com o algoritmo *H-SOFT*. Os experimentos poderiam ser conduzidos em cenários emulados, onde as simulações são muito mais abrangentes. Além disso, cenários equipados com conexões únicas, com fio, não representam os atuais

cenários, que utilizam diversos tipos de conexões sem fio.

PEREZ, K. G *et al* [49] também sugerem a redução do espaço de armazenamento das tabelas de fluxo, mas agora com a proposta de uma arquitetura de pesquisa de tabela de fluxo múltiplo, através da introdução de um filtro de tráfego *OpenFlow* para pesquisa multidimensional de tabelas de fluxo. A proposta se concentra em uma arquitetura de pesquisa de tabela de fluxo múltipla baseada em pesquisa de campo único paralelo. Entretanto, assim como ocorreu em [17], os experimentos poderiam se aproveitar dos cenários emulados. Além disso, os autores poderiam aplicar seus experimentos em ambientes atuais, tais como os vCPEs e as Sd-VANETS.

KIM, E. *et al* [35], abordam a implementação ineficiente do Encadeamento de Funções de Serviço, que vem do uso de uma única tabela de fluxo. Isso ocorre porque o número de entradas de fluxo na tabela de fluxo aumenta significativamente de acordo com o incremento do número de cadeias de serviço. Dessa forma, o *switch* deve procurar uma tabela de fluxo maior para processar pacotes, e isso também torna mais difícil atualizar o caminho das cadeias de serviço. Para resolver este problema, os autores propuseram esquema de construção de tabelas de fluxo modificado para implementar o encadeamento de funções de serviço com múltiplas tabelas de fluxo. Nesse esquema, as entradas de fluxo são inseridas na tabela de fluxo correspondente, classificadas como portas de entrada do *switch*, e então podem ser facilmente categorizadas e gerenciadas por meio de várias tabelas de fluxo. Fazendo isso, o *switch* pode pesquisar apenas a tabela de fluxo correspondente para processar os pacotes ou para atualizar o caminho das cadeias de serviço. Com isso, os autores afirmam ter conseguido reduzir a ineficiência da construção da tabela de fluxo para implantação do Encadeamento de Funções de Serviço no NFV. Embora os autores tenham conduzido experimentos realizando a diminuição do tamanho das tabelas de fluxo, não sugerem aplicações específicas onde essas reduções podem ser aplicadas.

3.2 vCPE

Nguyen, T. *et al* [46] concordam que o uso do paradigma NFV se beneficia das tecnologias de virtualização de TI para desacoplar funções de rede de *hardware* proprietário, para que possam ser executadas em *software* em plataforma de *hardware* comercial disponível. Afirmam que com o apoio do NFV, os provedores de serviços de comunicação podem fornecer soluções de equipamentos de configuração dos clientes virtualizados (vCPEs), que reduzem o número e o custo dos dispositivos de *hardware* físico necessários nas instalações do cliente, deixando-os capazes para hospedar conectividade e outros recursos de valor agregado. Com isso, os autores apresentam três abordagens de implementação na plataforma vCPE, ou seja, vCPE centralizada, vCPE distribuída e vCPE híbrida. Foi

realizado um experimento para medir o uso de recursos em diferentes modelos de implementação do vCPE.

Após a análise dos experimentos, foi constatado que cada um deles tem suas próprias vantagens e desvantagens em relação ao custo financeiro, à latência, ao uso de recursos, etc. Em termos de uso de recursos entre vCPE centralizado e vCPE distribuído, os resultados experimentais mostraram que o uso de recursos do equipamento do cliente em vCPE distribuído é significativamente maior do que na vCPE centralizada, enquanto o recurso do controlador central na vCPE centralizada é evidentemente mais consumido do que na vCPE distribuída.

Huang, N. *et al* [29] propõem um modelo de gerenciamento de múltiplas tabelas de fluxo para implementar funções de rede onde essas mesmas tabelas definem todo o processamento. Com o objetivo de substituir o *CPE* baseado em *hardware*, o artigo apresenta a criação de seis funções de rede virtuais, organizadas de uma forma que torne possível um gerenciamento eficiente do fluxo dos pacotes. São realizados experimentos para demonstrar a flexibilidade da nova estrutura diante da integração com sistemas de segurança (*Firewall, IPS e IDS*). Após a comparação da estrutura de tabela única com a de múltiplas tabelas foi possível avaliar que o desempenho da rede permaneceu semelhante, sem alterações que comprometam o seu bom funcionamento.

De acordo com os autores, o modelo de *switches* equipados com única tabela causaria restrições na implementação de suas funções de rede. Foram apontadas duas condições sob as quais uma única tabela de fluxo torna-se muito restritiva. A primeira é uma condição em que um único pacote deve executar ações independentes com base na correspondência com diferentes campos do *switch*. A segunda é uma condição em que o pacote requer processamento em dois estágios. Com esse entendimento, os autores argumentam que caso quisessem implementar duas funções de rede, como *QoS* e *firewall*, as regras necessárias para isso se tornariam muito complexas. A Figura 3.2 apresenta a ideia geral do experimento.

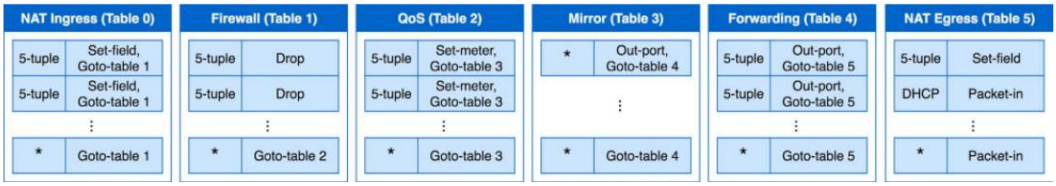

Figura 3.2: *Funções de rede implementados com múltiplas tabelas de fluxo. Extraído de* [29]

A Figura 3.2 apresenta o modelo proposto por [29]. Ao ser processado por uma tabela de fluxo, o pacote é comparado às entradas de fluxo da tabela do *switch* e adiciona a ação correspondente ao conjunto de instruções. Neste caso o pacote pode executar o

conjunto de instruções imediatamente ou executar após terminar sua jornada pelo *switch*. Com isso, o pacote é processado tabela por tabela em uma determinada seqüência. Como apresentado na Figura 3.2, as funções de rede dos serviços vCPE propostos foram o *Firewall, NAT, DHCP*, função de encaminhamento, espelhamento de tráfego e *QoS*.

Por fim, através da estrutura apresentada, os autores realizaram dois experimentos para avaliar o desempenho das estruturas com múltiplas tabelas. Foi utilizado a função *NAT* e o serviço de encaminhamento. Nesses experimentos foi constatado que o desempenho da estrutura com múltiplas tabelas é semelhante ao de tabela única.

3.3 VANETs definidas por *Software* - (SD-VANET)

A aplicação de SDN a VANETs é uma proposta que começou a ser aplicada em anos recentes. O primeiro trabalho foi apresentado por Ku, I. *et al* [38]. Desde então, várias pesquisas tem sido propostas para apresentar o grande potencial do paradigma SDN em melhorar o gerenciamento de recursos nas VANETs. Algumas dessas propostas podem ser conferidas em [22], [68], [62], [58] e [40].

Zheng, K. *et al* [68], defendem que em Sd-VANETs, um controlador primário tem como objetivo manter uma visão global da rede enquanto que outros controladores, implementados como controladores secundários, gerenciam as aplicações que tenham restrições de tempo real mais rígidas. Isso ocorre porque uma arquitetura de rede definida por *software* totalmente centralizada com um único controlador não consegue atender aos requisitos de baixa latência de futuras aplicações veiculares. De acordo com os autores, uma rede equipada com um único controlador pode não conseguir o controle em tempo real das aplicações e prejudicar todo o projeto da rede.

Nesse trabalho, os autores poderiam fazer uso da correta manipulação do *switch OpenFlow* e inserir as funções utilizando as múltiplas tabelas de fluxo.

Em [55], Salahuddin, M. A. *et al* oferecem outra solução para atender as aplicações que tenham restrições de tempo real mais rígidas. O compartilhamento de algumas tarefas com as estações base (*BSs*), deixaria o controlador atuando de maneira híbrida. Consequentemente, isso reduziria a sobrecarga incorrida no rastreamento das posições dos veículos arquitetados em uma Sd-VANET. Os autores não falam de soluções específicas onde sua implementação poderia ser aplicada. Existem muitas opções, tais como monitoramento das condições das estradas e rodovias, comunicação instantânea com outros veículos, etc. O compartilhamento de tarefas com o controlador administrando o *switch OpenFlow* com múltiplas tabelas deixaria essa aplicação muito mais robusta.

Chahal, M. *et al* [10] fazem um resumo sobre os desafios e aplicações possíveis das VANETs. Falam sobre vários casos e especificam algumas possibilidades de implantação de VANETs integrada com SDN. Dentre as soluções apresentadas estão a segurança

nas rodovias, monitoramento das condições das estradas e rodovias, *streaming* de vídeo, comunicação instantânea com outros veículos utilizada para o controle e gerenciamento de tráfego. Em seguida elaboram uma arquitetura de comunicação veicular definida por *software* (*Software-defined vehicular communication* - SDVC).

Não se apresentam aqui as consequências de se inserir um grande número de funções nos dispositivos utilizados para encaminhamento dos pacotes, tais como os *switches OpenFlow*. E a construção de equipamentos munidos de múltiplas tabelas de fluxo é uma solução pra esse tipo de problema.

Kaur, R. *et al* [33] apresentam alguns desafios à segurança das VANETs. Os autores afirmam que a troca de informações sem fio entre automóveis fazem com que as propriedades de confidencialidade, privacidade e autenticidade sejam violadas pelos invasores. Com isso, o artigo foca principalmente nos requisitos de segurança que devem ser cumpridos para proteger as VANETs. Neste caso, os autores poderiam realizar experimentos para demonstrar a ineficiência gerada pelas trocas de informações *wireless*. Mais uma vez destacamos a importância da aplicação de funções diversas em múltiplas tabelas de fluxo.

Vários outros autores também argumentam sobre questões relacionadas a segurança das VANETs. Por exemplo, Hussein, A. *et al* [30] apresentam um esquema geral e eficaz para implementar serviços de segurança em uma rede VANET baseado na integração com SDN em um ambiente *5G* sem sobrecarregar o controlador ou gerar atrasos nos dados ou no plano de controle. Um módulo de segurança foi implementado no controlador utilizando o *POX Controller*. Os autores mostraram como esse esquema pode resolver os importantes problemas de segurança que as Sd-VANETs tradicionais enfrentam. Um mecanismo de detecção e prevenção de ataques foi proposto com base em um novo plano de segurança. Aproveitando os benefícios do ambiente *5G* e da computação em névoa, o sistema provou atender aos requisitos da Sd-VANET de baixa latência e respostas mais altas. Além disso, foi apresentado um protótipo da arquitetura e dos mecanismos de segurança sugeridos, que por sua vez, foram implementados e testados em diferentes cenários para comprovar a eficácia da proposta.

Não se apresentou em [30] como a rede reagiria em situações de Sd-VANETs com alta latência. Nestes casos, o uso de múltiplas tabelas de fluxos nos *switches* trariam maior flexibilidade a rede.

A literatura fala com frequência dos benefícios de se utilizar SDN em aplicações VANETs. Uma importante aplicação no uso das Sd-VANETs é prestação de serviços de resgate. Martinez, F. J. *et al* [41] apresentam uma visão geral do estado da arte atual, discutem os projetos atuais, bem como seus objetivos, e destacam como os serviços de emergência e a segurança no trânsito irão evoluir com a combinação das redes de comunicação veicular e o transporte rodoviário inteligente. Os autores defendem que

as tecnologias contribuíram de maneira clara para a mudança no curso das ações e das medidas que deveriam ser tomadas após a ocorrência de um acidente; saindo de uma simples chamada de celular feita por uma testemunha, ao sistema de notificação de acidentes *eCall* em vigor na União Européia. Também defendem que num futuro próximo, os sistemas de notificação de acidentes serão especialmente projetados para serviços de resgate de pós-colisão. Combinando comunicações *V2V* e *V2I*, novos Sistemas de Transportes Inteligentes (*ITS*) surgirão com a capacidade de melhorar a capacidade de resposta dos serviços de emergência na estrada e com isso conseguirão permitir as seguintes soluções: (a) comunicação direta entre os veículos envolvidos no acidente, (b) entrega automática de dados relacionados ao acidente para a Unidade de Controle (UC); e (c) uma avaliação automática e preliminar dos danos com base na comunicação e no processamento de informações. E por fim, os autores informam que os futuros serviços de emergência baseados em *ITS* têm como objetivo atingir um baixo nível de fatalidades, melhorando significativamente o tempo de resposta e uso eficiente de recursos.

Todas as implementações sugeridas e apresentadas por [41], se beneficiariam grandemente com a utilização das múltiplas tabelas de fluxo.

Com um olhar diferenciado, Arif, M. *et al* [4] abordam os possíveis problemas que as VANETs poderão sofrer no futuro, e como a utilização da SDN pode ser vantajosa na construção de medidas de combate aos novos problemas. O trabalho destaca a utilização de um único controlador sem fio para gerenciar toda a VANET. Embora sejam abordados futuros problemas, os autores não mencionam como resolver as limitações do uso de estruturas equipadas com únicas tabelas de fluxo.

Dos Reis Fontes, R. *et al* [13], enumeram os atributos positivos das Sd-VANETs, e faz uma análise sobre a necessidade de repensar a abordagem SDN tradicional. Os autores realizam experimentos fazendo uso do emulador de redes Mininet-WiFi para simular a utilização de recursos sem fio e com fio em ambientes veiculares dinâmicos empregando pontos de vistas teóricos e práticos. Sua proposta visa trazer a programabilidade do paradigma SDN para dispositivos multi-interface a bordo de veículos.

3.4 Discussão Sobre os Trabalhos Relacionados

Nguyen, T. *et al* [46] apresentam três abordagens de implantação na plataforma vCPE, ou seja, vCPE centralizada, vCPE distribuída e vCPE híbrida. Foi realizado um experimento para medir o uso de recursos em diferentes modelos de implantação de vCPE. Após a análise dos experimentos foi constatado que cada um deles tem suas próprias vantagens e desvantagens em relação ao custo financeiro, latência, uso de recursos, etc. Embora tenha sido realizado testes comparando o desempenho das estruturas, nenhum modelo utilizando *switches* equipados com múltiplas tabelas de fluxo

foi proposto. O que, em um cenários de implantação de múltiplas funções ou serviços de rede deixaria o *switch OpenFlow* com maior capacidade de produção sem a limitação imposta pelo uso do modelo de única tabela.

Huang, N. *et al* [29] propuseram a criação de uma estrutura de rede com *switches* equipados com múltiplas tabelas de fluxo, e cada uma das tabelas armazenava uma função de rede especifica. As funções de *NAT* de entrada, *Firewall, QoS*, Espelhamento, Encaminhamento e *NAT* de saída foram instaladas e organizadas começando pela tabela 0 (zero) e finalizando na tabela 5 (cinco). Os experimentos apresentados provaram que quando o *switch OpenFlow* é configurado seguindo o modelo de múltiplas tabelas de fluxo, a estrutura da rede mantém um desempenho parecido com a estrutura de tabela única. Mas a principal vantagem do modelo de múltiplas tabelas é a flexibilidade na manipulação das funções. O que pode ser feito com o simples ato de indicar a tabela específica onde a referida função se encontra. O que de outra maneira precisaria da inserção e controle por parte do programador da rede, de um maior número de atribuições de prioridades para as funções.

Todos os experimentos realizados em [29] foram feitos em equipamentos físicos de arquitetura fechada, servidores *Bare Metal* de inquilino único, o que limita as possibilidades de outros tipos de testes e experimentos necessários para a comprovação da eficiência dos *switches OpenFlow*. No campo da pesquisa, o custo para a aquisição de equipamentos dedicados apenas para a realização de testes de desempenho é muito alto. O que, nas formas modernas de virtualização esse problema é resolvido, pois é possível realizar testes e experimentos em qualquer equipamento simples e de uso geral.

Tendo em vista o amplo campo de utilização dos *switches OpenFlow* dentro das SDNs, a realização de simulações em outros ambientes reais, tais como as VANETs, tornariam a sua implantação e o compartilhamento dos resultados obtidos impraticáveis devido o alto custo de aquisição de equipamentos dedicados. Sendo que esse tipo de problema é facilmente resolvido por meio da utilização dos emuladores de redes, tais como o Mininet e o Mininet-WiFi.

Chen, Z. *et al* [11] propuseram a implementação de várias sub-tabelas de fluxo dentro de uma única tabela principal. A ideia consiste em subdividir uma única tabela em vários outros sub-fluxos, o que, de acordo com os autores, otimiza o espaço de armazenamento da tabela no *switch*, gerando com isso um importante ganho de armazenamento, em especial nos equipamentos que possuem memória *TCAM*. As referidas vantagens foram constatadas após se realizar testes comparativos entre os equipamentos configurados com uma tabela de fluxo padrão e a tabela de fluxo otimizada com o modelo de sub-fluxos.

Apesar de o modelo sugerido por [11] realizar a diminuição dos fluxos nas memórias *TCAM*, os *switches OpenFlow* se beneficiariam pouco desse experimento, tendo em vista que os mesmos podem ser simulados e instanciados em ambientes

computacionais equipados com memórias mais baratas e simples, tais como as memórias *RAMs*.

Dos Reis Fontes, R. *et al* [13], tratam das dificuldades encontradas em gerenciar recursos em ambientes dinâmicos como os encontrados na Sd-VANET. Também fazem uma abordagem das aplicações e serviços fornecidos pelas Sd-VANETS e realiam uma aplicação do conceito como alternativa de programação aberta e conduzem experimentos baseados em cenários realísticos para homologar sua abordagem. Entretanto, seus experimentos poderiam oferecer maiores ganhos se a implementação das funções consideradas nos cenários tivessem sido realizadas com o esquema de múltiplas tabelas de fluxo.

Nos experimentos e trabalhos apresentados nas Seções 3.1, 3.2, e 3.3, foram relatadas soluções relacionadas à implementação de vCPEs equipados e se beneficiando do esquema de múltiplas tabelas de fluxo. Também foram enunciadas implementações relacionadas a Sd-VANETs mas sem a existência de múltiplas tabelas de fluxo para tornar ainda mais flexível suas aplicações. Quando aplicado às VANETs, o paradigma SDN requer extensões e modificações adequadas para enriquecer seu escopo além de seu design original, muito mais do que em outras redes sem fio [13]. Com isso, concluímos que qualquer aplicação moderna que busque se beneficiar da flexibilidade, programabilidade e controle centralizado, oferecido pela SDN, devem levar em consideração a correta prototipação do principal elemento da rede definida por *software*, o controlador. Fazer com que o controlador gerencie os *switches* e os deixem equipados com múltiplas tabelas de fluxo, garante uma rede mais flexível e eficaz. E esse é o principal objetivo da proposta deste trabalho. O Capítulo 4 considera em detalhes essa proposta.

CAPÍTULO 4

Proposta de Implementação de Múltiplas Tabelas de Fluxo (MTF) em *switches OpenFlow*

Este capítulo apresenta a proposta e detalhes sobre sua implementação. Em primeiro lugar, na Seção 4.1, é apresentado as limitações do *Switch OpenFlow*, implementado no esquema UTF (Única Tabela de Fluxo). Introduzimos a proposta de múltiplas tabelas na Seção 4.2. Em seguida, na Seção 4.3, é apresentado a arquitetura do esquema MTF (Múltiplas Tabelas de Fluxo). A partir de então as estruturas passam a ser abordadas simplesmente como UTF e MTF. Depois, na Seção 4.3.1 é feito uma análise detalhada sobre cada uma das funções e as ações envolvidas na proposta. Na Seção 4.4, é apresentado a arquitetura e os parâmetros do esquema UTF, que serve como base de apoio na comparação com o esquema MTF. Por último, na Seção 4.5 passa-se para a implementação do vCPE, e da Sd-VANET em 4.6, ambos instanciados e simulados pelo Mininet-WiFi.

De acordo com os trabalhos relacionados, apresentados no Capítulo 3, o estado da arte está bem avançado em assuntos relacionados as Redes definidas por *Software* (SDN) e suas consequentes extensões na área de virtualização dos equipamentos de configuração dos clientes (vCPEs) e Redes veiculares móveis sob demanda definidas por *software* (Sd-VANETs). E nesse contexto, o protocolo *OpenFlow* tem sido apresentado como o principal conjunto de normas e procedimentos que regulamenta esses novos paradigmas.

Na Seção 1.2, vimos que o desenho do *switch OpenFlow*, bem como de sua tabela de fluxo, passa a ser um dos principais desafios na busca do melhor desempenho e flexibilidade da SDN. Isso porque, nas redes baseadas no protocolo *OpenFlow* as estratégias de encaminhamento dos pacotes são determinadas por fluxo. E a definição dessas estratégias baseadas nas correspondências dos pacotes presentes nas entradas da tabela de fluxo, está se tornando cada vez mais complexo, devido a nova arquitetura da internet, com seus novos tipos de aplicativos e variados formatos de mídias [43]. Desse modo, a tabela de fluxo única da implementação padrão do *switch OpenFlow* pode levar ao rápido crescimento do espaço de armazenamento e, finalmente, causar o estouro da tabela; Além disso, a inserção de mais e mais funções deixariam os *scripts* muito longos

e tornariam sua manipulação muito complexa.

Existem vários trabalhos que apresentam os problemas implementados em uma única tabela de fluxo em *switches OpenFlow* [11], [17], [49], [35]. Como já foram apresentados, os problemas são que um único pacote deve executar ações independentes com base na correspondência com diferentes campos do *switch*, e o problema ocasionado pelo fato do pacote requerer processamento em dois estágios. De maneira geral isso gera o rápido crescimento do espaço de armazenamento e o consequente estouro da memória do *switch* [11]. Além disso, causa dificuldade em se estabelecer um encadeamento de funções de serviço devido ao aumento da tabela de fluxo, quando ela recebe um número maior de funções [17].

4.1 Limitações da implementação de Única Tabela de Fluxo (UTF) em *switches OpenFlow*

Na versão 1.0 do protocolo *OpenFlow*, o encaminhamento de pacotes dentro de um *switch* lógico *OpenFlow* é controlado por uma única tabela de fluxo que contém um conjunto de entradas de fluxo instaladas pelo controlador. As entradas de fluxo possuem campos de correspondência, ações e uma prioridade atribuída pelo controlador. O *switch* verifica os pacotes de entrada, nos campos de correspondência de entrada de fluxo. A entrada de fluxo de correspondência com maior prioridade define (através de suas ações), como lidar com os pacotes correspondentes. O conjunto de pacotes que correspondem a uma entrada é, por definição, um fluxo nos termos *OpenFlow*. Assim, o controlador, preenchendo a tabela de fluxo único com entradas de fluxo, define todos os fluxos relevantes e como eles devem ser manuseados [27].

No entanto, de acordo com [27], existem pelo menos duas situações onde um *switch* equipado com uma única tabela de fluxo pode causar limitações na rede. A primeira situação envolve a realização de ações independentes baseadas na correspondência de diferentes campos em um pacote. A segunda situação envolve um modelo natural de processamento em duas etapas. Nesse modelo de dois estágios, os pacotes são primeiro marcados (ou meta-marcados) com base em alguma característica do pacote, em seguida, ocorrem mais correspondência e processamento. Teoricamente, cada uma das situações pode ser tratada com o uso de uma única tabela, mas a necessidade de combinar campos correspondentes força uma explosão de entradas de fluxo na tabela. O ponto chave é que esses casos de uso podem ser tratados de forma simples adicionando mais tabelas de fluxo.

Uma das ações do *switch* ao receber um fluxo é verificar a compatibilidade do endereço *MAC* do pacote com seus respectivos campos, o que envolve procurar o endereço *MAC* de destino do pacote na tabela *MAC-address* do *switch*. Caso o *MAC* de destino

for encontrado, o pacote é encaminhado para a porta de entrada da tabela que contem o endereço *MAC*. Se o endereço de destino não for encontrado, o pacote é retransmitido para todas as portas (excluindo a porta de entrada). O processo de aprendizado de endereço *MAC* é comumente usado para preencher a tabela de endereço *MAC*. E esse aprendizado envolve procurar o endereço *MAC* de origem do pacote (normalmente com o *ID VLAN*) na tabela endereço *MAC*.

Quando o *MAC* de origem não for encontrado, então ele pode ser "aprendido", através de um processo local ou enviando o cabeçalho do pacote para que o controlador possa processá-lo. Embora, em teoria, tanto a função de encaminhamento quanto a função de aprendizagem possam ser feitas com uma única busca de dois endereços, esta abordagem resultaria em uma explosão de entradas. Por exemplo, para buscar N endereços *MAC* exigiriam entradas $O(n^2)$, de modo que 1.000 (mil) endereços *MAC* exigiriam 1.000.000 (um milhão) de entradas (além de todas as mensagens extras para gerenciar essas entradas) [26].

Entretanto, baseados nesse mesmo exemplo, usando duas tabelas de fluxo *OpenFlow*, seriam necessárias apenas entradas $2N$, utilizando o recurso de sincronização de tabela, disponíveis nas versões mais recentes do protocolo *OpenFlow* [26]. A Figura 4.1 apresenta a situação exemplificada aqui.

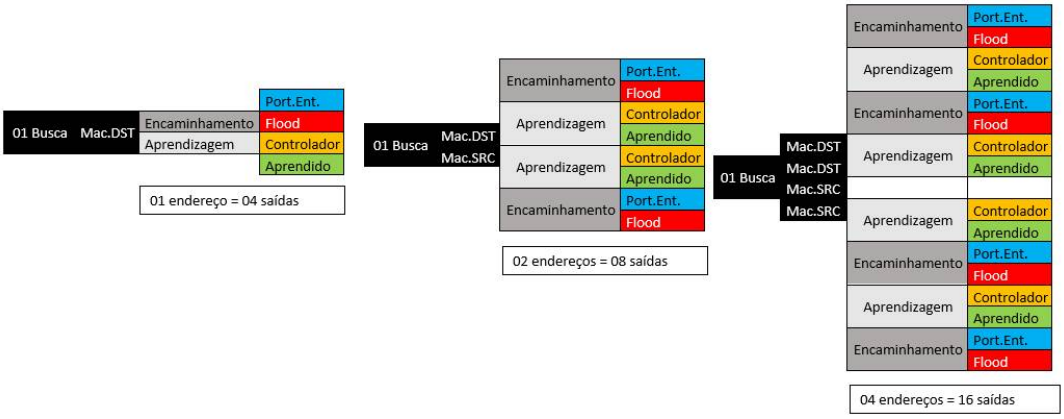

Figura 4.1: *Explosão de entradas de fluxo nas saídas do switch OpenFlow baseado em tabela de fluxo única.*

De acordo com a Figura 4.1 as informações que saem para as entradas da tabela vão aumentando gradativamente, ficando no mínimo o dobro e no máximo o quadrado da quantidade de endereços buscados, de modo que com a busca de 01 (um) endereço *MAC* são gerados 04 (quatro) entradas de dados. Similarmente, com a busca de 02 (dois) endereços *MAC* são gerados 08 (oito) entradas de dados, e com a busca de 04 (quatro) endereços *MAC* é gerado 16 (dezesseis) entradas de dados. Seguindo essa lógica, na

medida em que se aumenta a quantidade de endereços, os resultados das entradas nos *switches* sofre um crescimento assintótico na ordem de $O(n^2)$ [27] [26].

A partir da versão 1.1, o protocolo *OpenFlow* passou a comportar a utilização de múltiplas tabelas de fluxo para definir como os dispositivos conseguirão se defrontar com os fluxos de pacotes que chegam a eles. E isso está totalmente alinhado com o atual cenário de migração da rede, onde a quantidade de funções e serviços tem crescido bastante em resultado da expressiva utilização da internet [12]. A Figura 4.2 apresenta o fluxograma detalhando o fluxo do pacote através de um *switch* sem a especificação de MTF.

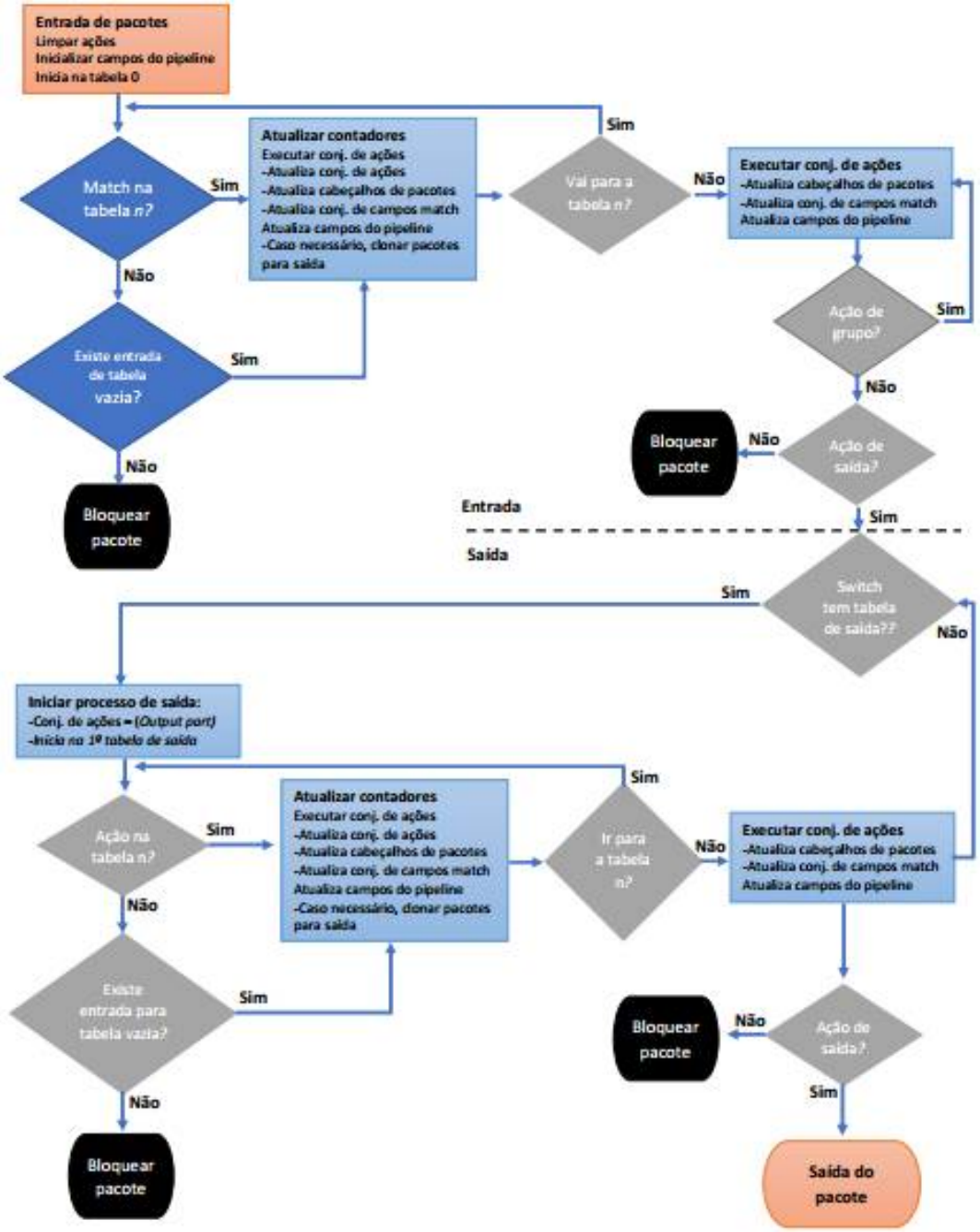

Figura 4.2: *Fluxograma simplificado detalhando o fluxo do pacote através de um switch OpenFlow. Adaptado de* [27]

De acordo com o fluxograma apresentado na Figura 4.2 cada entrada de fluxo contém um conjunto de instruções que são executadas quando um pacote corresponde à entrada do *switch*. E estas instruções podem resultar em alterações no pacote ou a um conjunto de ações e/ou processamento do *pipeline*.

Ao entrar no *switch* o pacote precisa executar a ação ***limpar ações*** para que as informações presentes no fluxo anterior não interfiram no fluxo atual. Em seguida é realizada a verificação de existência de correspondência em determinada tabela (Tabela n), se existir, um conjunto de instruções passa a ser executada. E se após a execução dessas ações o pacote for encaminhado para a próxima tabela o ciclo se repete. Quando o conjunto de instruções de uma entrada de fluxo não contém uma instrução "Ir para a próxima tabela", o processamento do *pipeline* pára e as ações no conjunto de ação do pacote são executadas. Além disso, caso não exista uma correspondência com determinada tabela, é verificado se existe no pacote entradas de tabela vazia. Caso exista, será executado o conjunto de ações correspondentes a essa tabela, caso o contrário, o pacote é descartado [27].

Em síntese observa-se no fluxograma que os pacotes são processados através do *pipeline*, onde os mesmos são combinados e processados na primeira tabela do *switch*, podendo também ser combinados e processados em outras tabelas. À medida que passa pelo *pipeline*, um pacote está associado a um conjunto de ações, acumulação de ações e um registro genérico de metadados. Esse conjunto de ação é resolvido no fim do *pipeline* e aplicado ao pacote. Os metadados podem ser combinados e escritos em cada tabela e permitem que o estado atual seja transportado entre as tabelas [27].

4.2 Proposta de Criação do esquema de Múltiplas Tabelas de Fluxo

Baseado no fluxograma apresentado na Figura 4.2, propomos a criação do esquema de múltiplas tabelas para distribuir as funções de rede, de modo a aumentar a flexibilidade e o desempenho da estrutura. A proposta é ilustrada através do fluxograma da Figura 4.3.

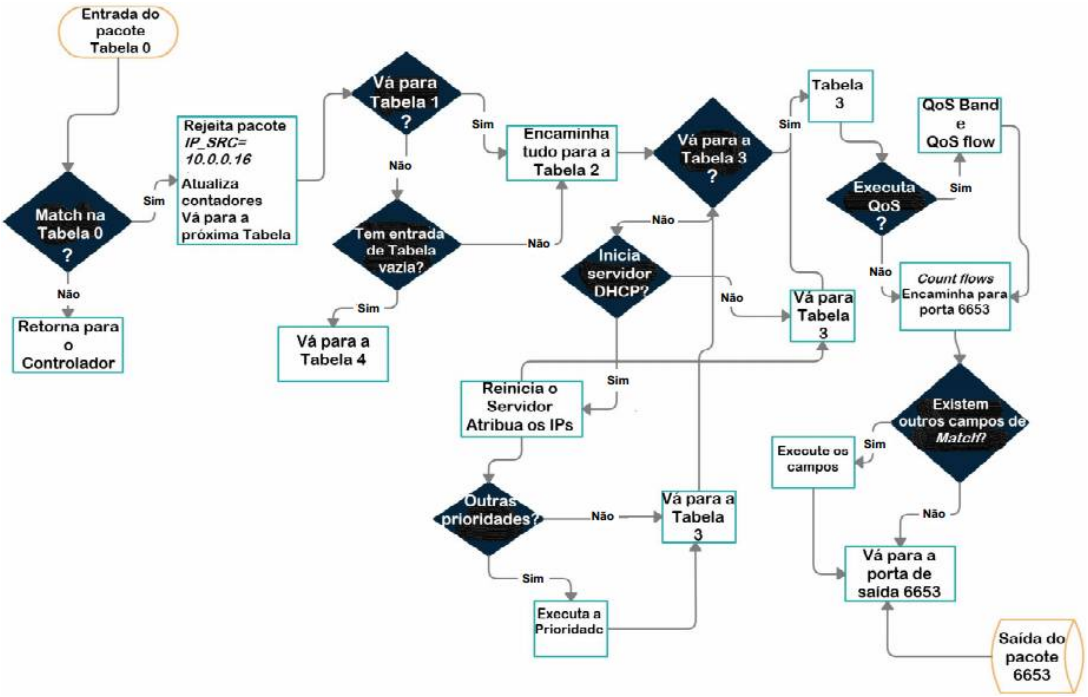

Figura 4.3: *Fluxograma detalhando a proposta de múltiplas tabelas com múltiplas funções*

De acordo com a Figura 4.3, quando o pacote entra no *switch*, através da tabela 0 (zero), dá-se início ao processo de busca de ações com suas respectivas prioridades. Na tabela 0, o pacote encontra correspondência com uma ação que rejeita todos os fluxos originados do endereço *IP 10.0.0.16*. Depois de atualizar os contadores de fluxo, o pacote é encaminhado para a próxima tabela, a tabela 1. Caso não encontre correspondência, o pacote retorna para o controlador.

Ao chegar à Tabela 1 o controlador precisa decidir se o pacote, de acordo com a prioridade definida para cada ação, segue caminho para a próxima tabela, ou se tem seus fluxos encaminhados diretamente para a última tabela, caso os fluxos atendam aos requisitos de tabela vazia (*Table miss*).

Na Tabela 2, o fluxo encontra a função *DHCP* com prioridade secundária, dando condições para que o pacote siga em frente, sem a necessidade de iniciar o servidor *DHCP*. Na proposta de múltiplas tabelas, a regra de encaminhar os fluxos para a próxima tabela, está definida com maior prioridade. Ao sair da Tabela 2, o pacote tem a opção de executar o serviço de *QoS* caso sua prioridade seja definida como alta. O que não ocorre em nossa proposta. A maior prioridade está sendo definida para a regra de encaminhar os fluxos para a porta de saída 6653.

Finalmente, o pacote termina o *pipeline* e é encaminhado para a porta de saída.

Outros detalhes sobre o percurso do pacote dentro da proposta de múltipla tabela, é apresentado a seguir, na Seção 4.3.

No Capítulo 2, foi visto que o protocolo *OpenFlow* permite a prototipação de comutadores equipados com múltiplas tabelas de fluxo. E esse esquema faz com que o controlador de redes RYU, gerencie de maneira eficaz, várias funções de rede capazes de proporcionar flexibilidade e interoperabilidade em diversas aplicações. O que não ocorre quando o *switch* é equipado com uma única tabela de fluxo. Neste caso o fluxo dos pacotes pode ser comprometido.

Entretanto, em um cenário que conta com o aumento da demanda por funções cada vez mais complexas, os programadores de *Switches OpenFlow* precisam organizar suas funções de uma maneira que seja possível produzir flexibilidade e ganho de desempenho simultaneamente. O que se consegue ao se instalar o modelo de funções organizadas e divididas em suas respectivas tabelas de fluxo (MTF).

Tendo em vista o estado da arte em assuntos relacionados à virtualização de funções de rede (NFV) em aplicações reais tais como vCPE e VANETs, surge a necessidade de atender as demandas cada vez mais crescentes de implementação de funções que possam interagir com outras funções ou dispositivos. E para tanto, o esquema de múltiplas tabelas no *Switch OpenFlow* atende a essa necessidade. Sendo assim, a correta programação do plano de dados em SDN é fundamental para o aumento da flexibilidade e o consequente aumento da interoperabilidade entre os sistemas de redes programáveis.

Para facilitar a prototipação e validação de novas aplicações de forma isolada, existem algumas ferramentas que simulam redes virtuais *OpenFlow*. O Mininet, foi a primeira ferramenta a possibilitar uma maneira rápida e fácil de testar aplicações *OpenFlow* [39]. Com a utilização de comutadores baseados em software, essa ferramenta oferece um ambiente emulado similar a um ambiente real com dispositivos físicos. Em sua versão mais recente, o Mininet-WiFi, traz algumas melhorias em relação ao primeiro Mininet [15]. Houve uma melhoria substancial de desempenho no mecanismo de emulação baseado em *container*, permitindo a criação de topologias de rede mais complexas, com centenas de *hosts* virtuais e dezenas de comutadores. Foi adicionada ao Mininet-WiFi a possibilidade de utilizar abstrações de software para executar totalmente um experimento de rede, por exemplo, criação da topologia da rede virtual, injeção de um tráfego de pacotes, modificação de parâmetros de interesse como atraso ou perda de pacotes e extração dos resultados do experimento.

Embora existam várias tecnologias provendo o serviço sem fio, o *WiFi* se destaca, principalmente pela boa relação entre custo e desempenho. E para esta tecnologia, o *IEEE 802.11* é o padrão para redes sem fio mais aceito mundialmente. No modo de infraestrutura, a arquitetura de uma rede 802.11 é composta principalmente por um ponto de acesso e estações sem fio. Já no modo de operação *ad hoc*, a arquitetura é composta

apenas por estações sem fio independentes.

4.3 Arquitetura Proposta da Implementação MTF

A Seção 1.2 apresentou a proposta de funções distribuídas em múltiplas tabelas de fluxo. Agora, na seção atual, apresentamos a arquitetura das estruturas de rede sem fio capazes de fornecer os serviços de Encaminhamento de pacotes, *Firewall*, *DHCP* e *QoS*, exemplificado no fluxograma da Figura 4.3. Embora o objetivo deste trabalho não seja avaliar especificamente essas funções, elas são corretamente implementadas e disponibilizadas, de modo a tornar o ambiente de testes bem próximo do ambiente real. Sendo, portanto, o desempenho geral da estrutura, com a presença de tais funções, o objetivo específico deste trabalho. Dessa forma, os nós simulados são capazes de enviar pacotes de dados, e esperar que os mesmos trafeguem através das funções distribuídas nas tabelas de fluxo, e possam assim, ter o seu desempenho medido e avaliado ao se comparar os ambientes equipados com única e com múltiplas tabelas de fluxo no *switch OpenFlow*.

Os cenários utilizados para aplicar as funções de rede são o vCPE e a Sd-VANET. Para o vCPE, apenas o emulador Mininet-WiFi é necessário. Já para o cenário Sd-VANET, além do Mininet-WiFi, para simular a mobilidade e definir o posicionamento dos nós da rede, é preciso utilizar o sistema de simulação de redes veiculares SUMO, pois ele faz a integração de toda a rede com o Mininet-WiFi [5]. A Figura 4.4 apresenta a ideia geral do esquema proposto. O mesmo esquema é utilizado para simular tanto o cenário vCPE, quanto o cenário Sd-VANET. A figura apresenta os principais componentes de um pacote de dados e o seu uso dentro das funções de rede, durante seu *pipeline*, ou caminho, que o mesmo pode percorrer, antes de ser executado.

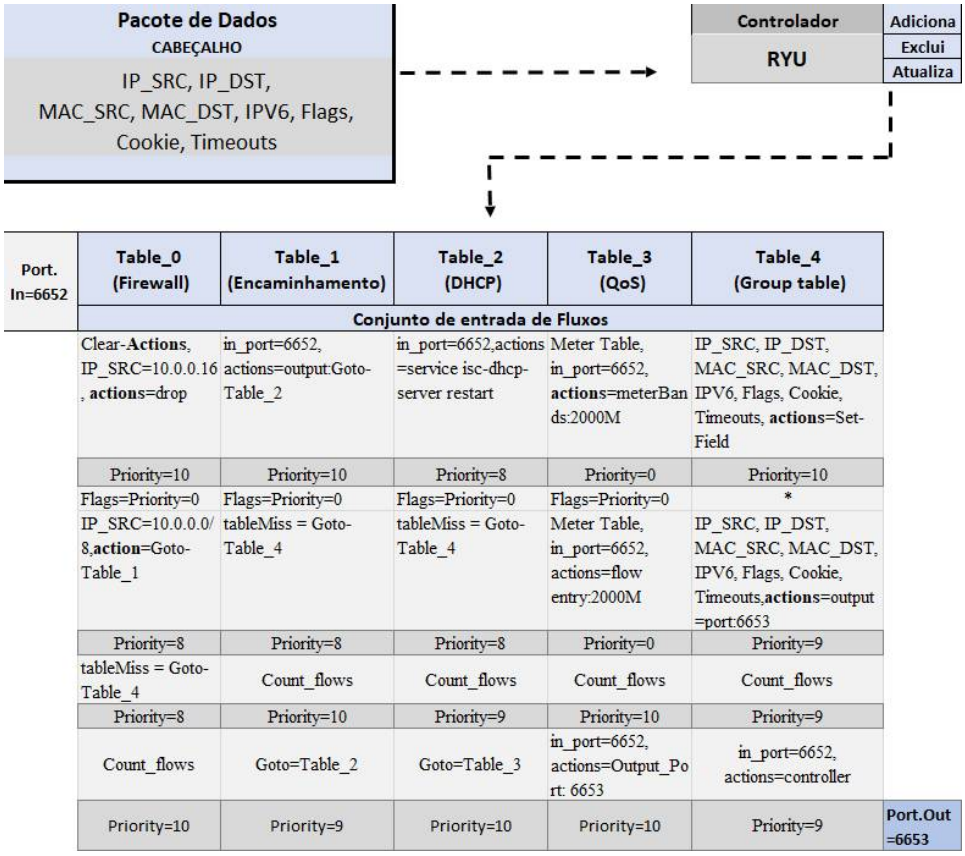

Figura 4.4: *Arquitetura proposta do esquema de Múltiplas Tabelas de Fluxo - (MTF)*

4.3.1 Implementação de Funções de Redes Virtualizadas

A Figura 4.4 apresenta as funções de rede virtualizadas implementadas. Cada função está associada a uma tabela de fluxo, sendo 05 (cinco) tabelas de fluxos: *Table-0 (Firewall), Table-1 (Encaminhamento), Table-2 (DHCP), Table-3 (QoS) e Table-4 (Group table)*. A última tabela está equipada com um conjunto de campos de correspondência extras, que tem a capacidade de resolver a correspondência dos pacotes, para que estes não precisem retornar ao controlador em casos de não correspondência. Portanto, a tabela *Table-4* não possui uma única função associada a ela, mas está equipada com um conjunto de ações necessária para o funcionamento adequada do esquema de múltiplas tabelas de fluxo. Com isso o pacote deverá ser comparado primeiro com as entradas de fluxo da tabela 0 (zero). Ao chegar ao controlador, o pacote é encaminhado para a porta de entrada do *switch*, que no caso em questão, é representada pelo número 6652. A seguir, fazemos uma análise detalhada de cada uma das tabelas e funções presentes na proposta MTF.

Tabela 0 - *Firewall*

Um *Firewall* atua como uma defesa contra diversos tipos de ameaças dentro da rede. E uma das estratégias de defesa é a ação de bloquear a execução de determinado fluxo. Por isso, a primeira tabela possui, como maior prioridade (*Priority=10*), a ação de rejeitar todos os pacotes originados do endereço *IP* com final 16 (dezesseis). Dessa forma, todos os pacotes seguirão caminho, menos o apontado com a ação *drop*. Em seguida, o pacote continuará pesquisando por ações de maior prioridade, e encontrará a ação de contar os fluxos (*Count-flows*), também com prioridade 10. O resultado dessa contagem poderá ser aproveitado para definir, por exemplo, a largura de banda utilizada pela função de *QoS*, presente na tabela de fluxo 3 (três).

Na sequência, o pacote encontrará as ações de direcionar os pacotes para a próxima tabela (*table-1*), com prioridade 9, e a ação de tabela vazia (*Table Miss*), com prioridade 8 (oito). Como o pacote já encontrou a correspondência de rejeitar um determinado endereço (*IP 10.0.0.16*), contar os fluxos e direcionar todos os demais fluxos para a próxima tabela (*Table-1*), não houve necessidade de executar a ação de tabela vazia (*Table Miss*), que é executada quando o pacote não encontra correspondência com nenhum outro campo.

Tabela 1 - Encaminhamento

Encaminhamento ou roteamento de pacotes (*Routing*) é o processo de escolha de caminhos para se enviar um pacote. No caso da Tabela-1 da proposta, o encaminhamento é feito de forma estática. Isso significa que o comando *in-port=6652*, enviará todos os pacotes originados da porta 6652 diretamente para a Tabela-2. Mas antes de ocorrer essa ação, o pacote irá verificar se não existe alguma outra ação com prioridade mais alta ou equivalente. E a ação *Count-Flows* deve ser executada juntamente com a ação de encaminhar tudo, por ter o número 10 como prioridade. A ação *Flags* ou sinalizadores, alteram a forma como as entradas de fluxo são gerenciadas, por exemplo, o sinalizador *OFPFF-SEND-FLOW-REM* aciona mensagens de fluxo previamente removidos para essa entrada de fluxo. Como sua prioridade é a menor, não será executada neste caso.

Após a contagem e registro dos fluxos, a ação de enviar o pacote para a Tabela-2 será executada. Caso isso não ocorra por algum motivo, o pacote será enviado diretamente para a Tabela-4. O que também ocorrerá se nenhuma das ações com maior prioridade seja executada, fazendo com que a ação *Table Miss*, com prioridade 8, finalmente ocorra.

Tabela 2 - *DHCP*

O protocolo *DHCP* é um serviço muito importante dentro de uma rede infraestruturada. Ele é o responsável pela distribuição de endereços *IPs* de maneira dinâmica,

sem precisar interferir nas configurações da máquina para se conectar à rede. Geralmente é possível atribuir *IPs* de três formas distintas. O método mais comum é a distribuição de forma dinâmica. Neste caso, quem oferece as configurações da rede é um servidor *DHCP*. Uma desvantagem desse método é que os endereços atribuídos serão trocados logo após a máquina cliente ser reiniciada ou atualizada.

O outro método é através da configuração automática, que apesar de parecer com o método anterior, irá fornecer os mesmos endereços *IPs*. O que é bem diferente do terceiro método, o Manual, onde o administrador da rede vincula o endereço *MAC* do equipamento cliente a um endereço *IP* específico. E esse último foi o método adotado para equipar uma das tabelas de nossa proposta.

Para atribuir endereços *IPs* dentro da estrutura de múltiplas tabelas, é preciso configurar um servidor *DHCP* e deixá-lo disponível para ser "chamado"a qualquer momento. Por esse motivo, o servidor *isc-dhcp-server* foi instalado. E logo no início da Tabela-2, a regra que faz com que o servidor entre em execução está definido com a prioridade 8. Ou seja, poderá ser executado caso não exista outra ação com maior prioridade. Nesse caso, a ação de enviar os fluxos para a Tabela-3 será executada primeiro, por ter a maior prioridade, indicada pelo número 10.

Neste caso, embora o servidor *DHCP* não seja executado nos experimentos com vCPE e Sd-VANETs, o mesmo estará disponível, bastando para isso que se atribua um número de prioridade maior na ação de reiniciar o serviço *DHCP*.

Tabela 3 - *QoS*

A Função *QoS* faz com que seja possível definir prioridades de acesso entre os fluxos e até mesmo melhorar a qualidade da conexão. As regras do *QoS* dentro do *OpenFlow* são tratadas como *meter tables*. Uma tabela de medidores consiste em entradas de medidores, definindo medidores por fluxo. Medidores por fluxo habilitam o *OpenFlow* para implementar a limitação de taxa, que é uma operação de QoS simples capaz de restringir um conjunto de fluxos a uma largura de banda previamente escolhida. Os medidores por fluxo também podem permitir que o *OpenFlow* implemente operações de policiamento de *QoS* mais complexas, como medição baseada em *DSCP* (*Differentiated Services Code Point*) que pode classificar um conjunto de pacotes em várias categorias com base em sua taxa. Medidores são totalmente independentes de filas por porta, no entanto, em muitos casos, esses dois recursos podem ser combinado para implementar estruturas de *QoS* de conservação de trabalho complexo, como *DiffServ* (Serviços Diferenciados) [44].

Na Tabela-3, a ação *Count Flows* será executada juntamente com a ação de enviar o pacote para a porta de saída 6653. Essa ação fará com que o percurso do pacote seja reduzido, e com isso, se obtenha um ganho de desempenho em comparação com a estrutura de única tabela. Opcionalmente, o pacote poderá ser enviado para a próxima

tabela (Tabela 4), caso os fluxos não tenham obtido correspondência nos outros campos da Tabela 3 (*Table-3*).

Vale destacar que, as ações de limitar a quantidade de pacotes a 2 MB, estão presentes na tabela, porém, com um número de prioridade menor. O que permitirá que o pacote execute somente as ações com maior prioridade, e siga seu fluxo livremente em direção ao destino final, que é a porta 6653. Muito embora a ação de dividir a largura de banda da rede, esteja presente na Tabela-3, essa ação será realizada com o próprio aplicativo gerador de tráfego, *iperf*, que limitará a largura de banda a 2 *MB*.

Tabela 4 - *Group Table*

Group Table ou tabela de grupo consiste na existência de várias outras entradas de grupo. Ou seja, fornece a estrutura a capacidade de uma entrada de fluxo apontar para um grupo de outras entradas, não especificadas nas tabelas anteriores, e que fazem com que, de alguma maneira o pacote seja correspondido, diminuindo a necessidade de o mesmo ser redirecionado para o controlador, que é outra vantagem da estrutura de múltiplas tabelas (MTF).

Normalmente, uma tabela de grupo possui ações que modificam os pacotes e os encaminham para uma porta de saída. Poderá também ter uma ação de grupo ou *group actions*, capaz de invocar outro grupo, e assim fazer com que o pacote necessariamente siga sua trajetória (*pipeline*) corretamente [44].

4.4 Implementação UTF *vs.* MTF

Com a proposta de múltiplas tabelas de fluxo (MTF) em *switches OpenFlow*, as aplicações tendem a obter um maior ganho de flexibilidade e desempenho. As diversas funções que podem existir em uma determinada aplicação, podem ser acionadas ou não, dependendo do que se espera delas. Mesmo que essas funções estejam presentes no *switch*, seu uso ficará condicionado a uma prioridade previamente atribuída. Isso pode indicar que uma função com prioridade baixa não será executada caso uma ação que modifique o processamento do *pipeline* seja executada. A Figura 4.5 apresenta a estrutura de Única Tabela de Fluxo (UTF), equipadas com as mesmas funções da estrutura MTF.

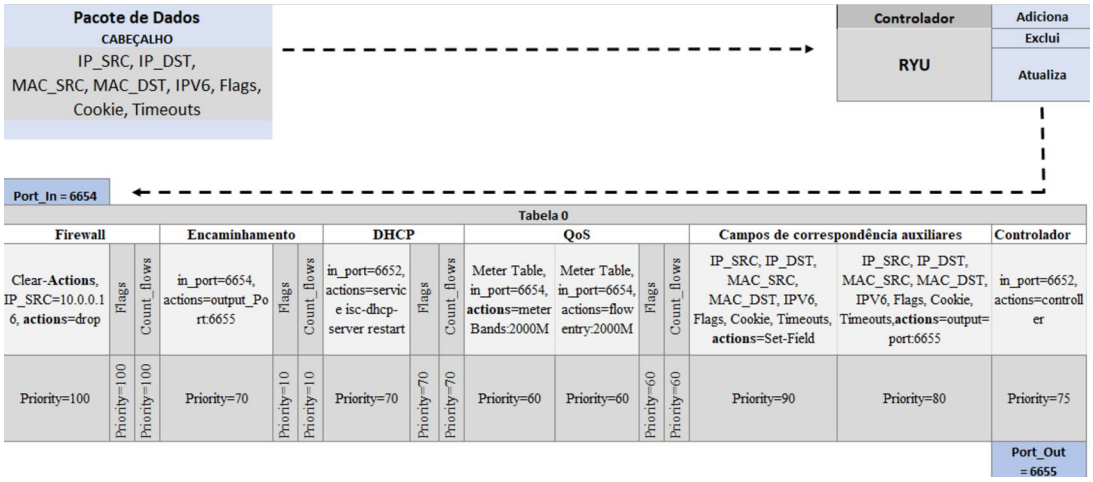

Figura 4.5: *Quatro funções distintas equipando uma UTF*

A Figura 4.5 apresenta a abstração do *switch OpenFlow*, instanciada com quatro funções de rede, e distribuídas em uma única tabela de fluxo. Esse esquema segue a mesma proposta de criação de múltiplas funções de rede, administradas pelo controlador RYU. A principal diferença é que na estrutura de múltiplas tabelas de fluxo, o pacote de dados chegará a seu destino com um menor número de saltos do que no esquema de única tabela de fluxo. No esquema de múltipla tabela o pacote executa somente quatro saltos, pois não precisará passar pela última tabela (tabela 4), para atingir a porta de saída. Existe uma regra, com uma maior prioridade, sendo executado primeiro. Já no esquema de única tabela, o pacote precisa executar todos os saltos antes de atingir seu alvo. Isso proporciona um ganho de 5% no desempenho total da estrutura.

Conforme se observa na Figura 4.5, o pacote entra na tabela de fluxo única e começa sua busca por ações que serão executadas de acordo com sua prioridade. A ação de rejeitar os fluxos originados do endereço *IP 10.0.0.16*, está indicada na tabela como a ação de maior prioridade (*Priority=100*), e que, portanto, será executado primeiro. Entretanto, o pacote não terá terminado sua busca por instruções. Ele primeiro precisará percorrer toda a tabela, para que em seguida comece a executar as ações determinadas por ordem de prioridade.

Ainda de acordo com a Figura 4.5, as ações estão distribuídas aleatoriamente, o que não impede que o pacote execute sua busca, pois ele precisa localizar e selecionar a ação de maior prioridade. Seguindo então esse princípio, a ação de rejeitar o pacote será executada em primeiro lugar. Em seguida, com prioridade 90, a ação de encontrar outros campos de correspondência, baseada em IP de origem (*IP-SRC*), *IP* de destino (*IP-DST*), endereço *MAC* de origem e destino (*MAC-SRC e MAC-DST*), protocolo *IPv6*, dentre outros, permitirá que o pacote atualize seu cabeçalho por meio da ação *Set-Field*.

A próxima ação, de prioridade 80 (*Priority=80*), direciona os fluxos, agora já atualizados, para a porta de saída 6655. Caso esse direcionamento não ocorra, a ação de prioridade 75 (*Priority=75*) devolverá todos os fluxos para o controlador decidir o que fará com ele. Caso isso não ocorra, as ações de prioridade 70 (*Priority=70*), irão atribuir endereços *IPs* através da ativação do servidor *DHCP*, e designará os fluxos originados da porta 6654, para a porta de saída 6655.

Os fluxos com prioridade igual a 60 (*Priority=60*), assim como ocorre na estrutura de múltiplas tabelas, estará presente, contudo não será executado nesse experimento. E por fim, os fluxos serão contabilizados, marcados com suas respectivas *Flags*, e encaminhados para seu destino final.

A estrutura equipada com UTF não possui a instrução ***GoTo-Table***, que faz com que os fluxos paralisem sua busca por ações e sejam encaminhados imediatamente para uma próxima tabela. Com a instrução ***GoTo-Table*** o pacote não precisará buscar outras ações dentro de uma tabela, pois essa instrução se sobrepõe as demais. Como resultado disso, os pacotes chegará a seu destino com um menor número de comparações e consequentes menores saltos.

As estruturas UTF e MTF serão instanciadas com duas aplicações importantes no contexto de NFV. A primeira propõe implementar um vCPE capaz de fornecer funções de *Firewall*, Encaminhamento, *DHCP* e *QoS*, que serão avaliadas utilizando as estruturas UTF e MTF. A segunda propõe implementar uma VANET definida por software (Sd-VANET), capaz de fornecer as mesmas funções do vCPE, a clientes que já se comunicam entre si, mas que precisam de funções auxiliares, disponibilizadas através de um dispositivo centralizado. Esta aplicação também será avaliada utilizando as estruturas UTF e MTF.

4.5 Cenário vCPE Proposto

No vCPE, as funções de rede devem estar disponíveis para serem utilizadas a qualquer momento. Por esse motivo, o controlador disponibilizará as quatro funções, tanto na estrutura UTF quanto na MTF. No vCPE, o *Switch OpenFlow* estará se comunicando com um controlador externo, que será o administrador de toda a aplicação. Os nós da rede irão representar os clientes, com seus diversos tipos de dispositivos, conectados a rede. Desta forma, o cenário onde um dispositivo, sendo capaz de fornecer serviços e funções para usuários domésticos conectados a ele, estará estabelecido. Criando assim um CPE virtualizado, avaliado com a utilização das estruturas UTF e MTF. A Figura 4.6 apresenta a topologia da proposta vCPE.

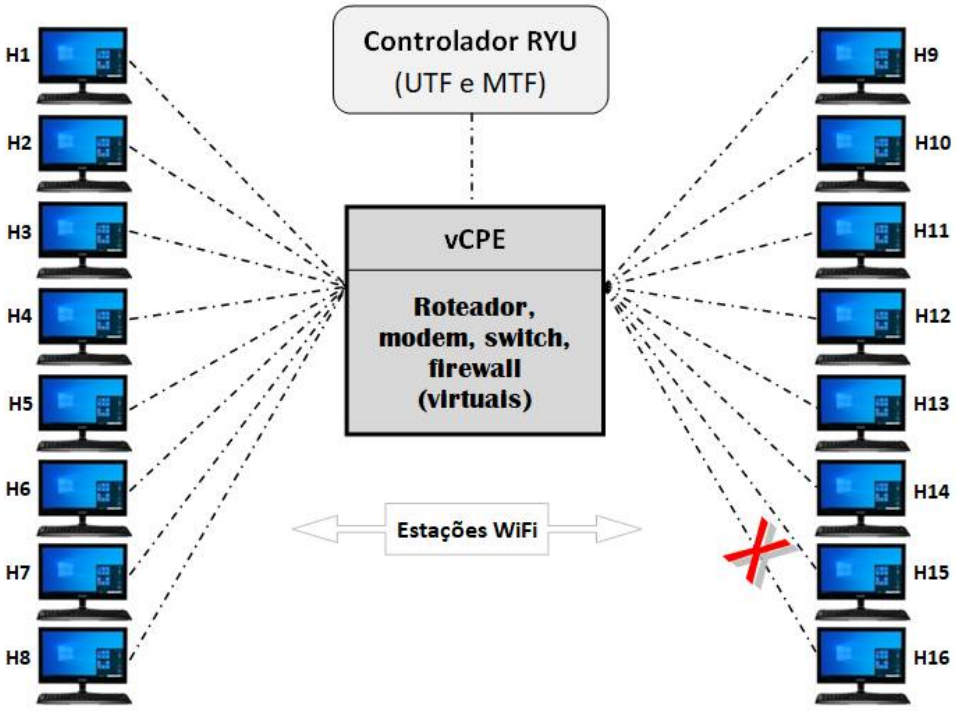

Figura 4.6: *Topologia e funcionamento da estrutura vCPE*

A Figura 4.6 apresenta o cenário vCPE que está composto de 16 *hosts* e um vCPE (Roteador, *switch, modem* ou *firewall virtuais*). De acordo com a Figura 4.6, o vCPE é controlado pelo aplicativo RYU. Internamente o vCPE está equipado com as estruturas de tabelas UTF e MTF, que armazenam funções de rede, organizadas por prioridade. Uma das funções tem como maior prioridade a ação de rejeitar os pacotes originados do endereço *IP 10.0.0.16*, conforme indicado pelo *X* vermelho sob a conexão da estação 16 (*H16*).

4.6 Cenário Sd-VANET Proposto

Na Sd-VANET as funções devem estar disponíveis para serem utilizadas a qualquer momento pelos dispositivos que estiverem dentro do raio de abrangência da rede. O elemento que recebe e centraliza as funções, representando o vCPE, equivale ao RSUs encontrados nas VANETs tradicionais. Esse RSU estará conectado as estações sem fio, que nesse cenário é tratado basicamente como carros. A Figura 4.7 apresenta a arquitetura e a representação gráfica da estrutura Sd-VANET.

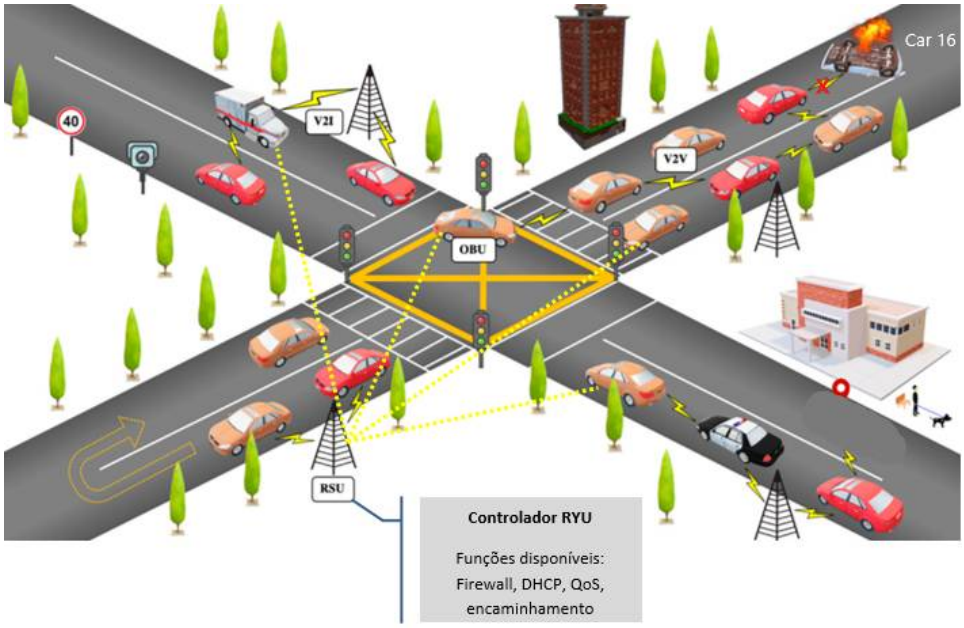

Figura 4.7: *Topologia e funcionamento da estrutura vCPE*

Conforme pode ser observado na Figura 4.7, o controlador RYU, equipado com UTF e MTF está ao alcance de todos os carros, que por sua vez estão se comunicando entre si. Entretanto o carro 16 (*Car 16*) não poderá se comunicar com a estrutura, pois ele está danificado devido a um acidente. Outros detalhes sobre a comunicação das estações serão tratados na próxima seção.

4.7 Conclusão

Neste capítulo apresentamos a proposta de simulação das duas aplicações, vCPE e Sd-VANET instanciadas com o emulador de redes Mininet-WiFi. Foi apresentada a arquitetura do esquema de múltiplas tabelas de fluxo (MTF), comparado com o esquema de única tabela (UTF). As duas arquiteturas possuindo quatro funções de rede, definidas com prioridades diferentes, o que inclusive, faz com que a execução das funções sejam seletivas. E por último, apresentamos a topologia das aplicações vCPE e Sd-VANET, destacando a execução de uma regra de *Firewall*, impedindo o fluxo de dados por endereço *IP*.

Para emular a Sd-VANET utilizamos o aplicativo de simulação de mobilidade urbana SUMO [5]. Através deste aplicativo é possível simular todos os parâmetros de mobilidade necessários para a definição do cenário Sd-VANET. Entretanto, o controle, os testes e as definições de rede são executadas pelo Mininet-WiFi [15].

A seguir, o Capítulo 5 apresenta a metodologia, os dados e os detalhes dos dois cenários apresentados aqui. Também se apresenta uma representação gráfica dos resultados dos experimentos utilizando as estruturas UTF e MTF.

CAPÍTULO 5

Avaliação da Proposta MTF e Resultados Obtidos

Neste capítulo são avaliados os objetivos propostos resultantes da implementação das aplicações vCPE e Sd-VANET equipadas com diferentes funções distribuídas em múltiplas tabelas de fluxo. A avaliação é feita por simulações para demonstrar os benefícios da proposta e o aprimoramento dos níveis de flexibilidade e dinamicidade alcançada pelas múltiplas tabelas de fluxo nos *switches OpenFlow*. São avaliadas as perdas de pacotes durante a transmissão de diferentes quantidades de fluxos, a variação do atraso na entrega de pacotes (*jitter*) e a quantidade de dados transferidos na rede (*throughput*). Com isso, é possível dimensionar o impacto causado no desempenho geral da rede em diferentes quantidades de nós e aplicações.

As seções estão organizadas da seguinte forma: Seção 5.1 apresenta a metodologia e as características dos dispositivos e do ambiente de simulação; Seção 5.1.1 traz os critérios que foram utilizados para a análise dos dados; Seção 5.2 descreve as aplicações que compõem o cenário de aplicação vCPE e suas características; Em 5.2.1 se executa os experimentos e tem seus resultados exibidos nas Seções 5.2.2, 5.2.3 e 5.2.4. Em seguida apresentamos o cenário Sd-VANET na Seção 5.3 e a execução do experimento na Seção 5.3.1. Por fim, nas Seções 5.3.2, 5.3.3 e 5.3.4 apresentamos os resultados através de gráficos comparando as estruturas UTF e MTF, onde é feito uma discussão na Seção 5.3.5 do que pode ser extraído com a aplicação de múltiplas tabelas de fluxo, proposta e desenvolvida neste trabalho para a melhoria e a extensão das aplicações vCPE e Sd-VANETS.

Com isso, apresentamos os experimentos usados para realizar a comparação das estruturas SDN equipadas com *switches OpenFlow* configurados com única tabela e múltiplas tabelas de fluxo. Estas comparações tem por base os conceitos apontados por [29], onde se destacou a flexibilidade como sendo uma das principais vantagens do uso de múltiplas tabelas de fluxo.

Foram estabelecidos dois cenários. O primeiro tem o objetivo de apontar a flexibilidade que o uso de múltiplas tabelas proporcionam na inserção de funções de rede em uma estrutura *vCPE* que busca oferecer serviços de acordo com o que é determinado

pelo cliente. E a organização dessas funções, tendo os *switches* caracterizados com múltiplas tabelas, diminui a quantidade de entradas de fluxo.

O segundo cenário emprega o conceito de VANETs junto a SDN, ou seja, Sd-VANETs. Como o ato de proporcionar flexibilidade também é uma das principais características das VANETs, busca-se nesse experimento apresentar o uso da estrutura de múltiplas tabelas de fluxo para implementar as funções de encaminhamento e segurança, que são funções de rede amplamente utilizadas na comunicação entre os veículos. Em ambos os casos, são realizadas comparações entre as estruturas caracterizadas com única e com múltiplas tabelas de fluxo.

5.1 Metodologia

As estruturas são emuladas e os experimentos são produzidos na plataforma de virtualização *VirtualBox*, versão 6.1. O Mininet-WiFi, é utilizado como o emulador para os cenários de rede sem fio, necessários para simular o *vCPE* e a VANET [15]. As redes são configuradas via linha de comando, utilizando classes da linguagem de programação *Python*, o que tornou a escrita e a leitura dos códigos relativamente fáceis. No que se refere ao cenário SDN, foi utilizado o controlador RYU devido a sua consolidação junto a academia, principalmente na realização de pesquisas na área de redes de computadores. O uso deste controlador possibilita uma curva de aprendizado muito boa pelo fato de possuir uma vasta documentação e uma comunidade bem ativa [54]. O *software* utilizado para testar as métricas de largura de banda, *jitter* e a vazão obtidos após a construção dos cenários propostos é o *Iperf* (versão 2.0).

Todos esses softwares estão sendo executados em uma máquina Dell Vostro 3550, equipada com um processador de 08 núcleos *Intel i7* de 2º geração, possuindo *8 Giga Bytes* de memória *RAM* e uma unidade de armazenamento *SSD* de *240 Giga Bytes*.

5.1.1 Tipo de Dados

As redes modernas estão sendo cada vez mais usadas para a transmissão de dados de aplicações multimídia, como, imagens, sons e vídeo. Para a transmissão desses dados o principal protocolo utilizado é o *UDP* (*User Datagram Protocol*) o que permite a transferência de datagramas em rede de telecomunicações sem o estabelecimento prévio de conexão. Isso se deve ao conteúdo do datagrama, que já contém uma quantidade suficiente de informações em seus campos de título. Além disso, o UDP não possui controle de fluxo, o que significa que a ordem dos pacotes transmitidos e recebidos pode mudar. Além disso, o remetente não tem nenhum meio de garantir que o pacote UDP tenha chegado à extremidade receptora corretamente, porque o UDP não inclui mecanismos de

confirmação. Um dos casos de uso típicos do UDP é o *streaming* de vídeo e áudio em tempo real. UDP é, portanto, um protocolo que contém um mínimo de complexidade entre os níveis de rede e protocolo de aplicativos. O protocolo UDP é especialmente útil nas situações em que a taxa de bits é mais importante do que a garantia de qualidade de serviço, por exemplo, no caso de áudio e vídeo [48]. Por esse motivo, com o objetivo de analisar o comportamento da rede, em situações onde se priorize o seu desempenho, será utilizado o protocolo UDP nos experimentos vCPE e Sd-VANETs.

Tanto no vCPE quanto na Sd-VANET, existe a possibilidade de a comunicação dos nós serem realizadas com mobilidade. No entanto, para a realização da comunicação dos dispositivos e a verificação do seu comportamento utilizando estruturas equipadas com única e múltipla tabela, os nós, neste trabalho, estarão estáticos, ou seja, com distâncias pré-definidas.

Com isso, o protocolo UDP é utilizado para transmitir dados estáticos, originados de um nó da rede, definido como servidor. No caso do vCPE, conforme apresentado na Figura 5.3, a estação 01 (*STA1*) é o servidor, que recebe os pacotes originados pelas estações 02 até a 16 (*sta 02 à sta 16*). O meio de comunicação é sem fio, e os pacotes percorrerão tanto o *switch* equipado com única tabela, quanto o equipado com múltipla tabela, seguindo as regras pré-definidas pelo controlador RYU.

No cenário Sd-VANET, conforme apresentado na Figura 5.14 as estações são chamadas de *CARs*, e o dispositivo da rede que atua como *access point*, é chamado de RSU (*Road Street Unit*), que recebe os pacotes *UDPs* originados nas estações. Neste cenário, assim como no vCPE, as estruturas com única e com múltiplas tabelas são aplicadas utilizando o controlador RYU.

Tanto no cenário emulado vCPE quanto no cenário Sd-VANET ficou estabelecido que o *host* com *IP* terminado em 16 não deve se comunicar com os demais *hosts* da rede. Isso é possível através da aplicação de uma regra de bloqueio disponível no protocolo *OpenFlow*. A Figura 5.1 apresenta um trecho do código, com a regra de bloqueio (*DROP*), presente nas duas estruturas.

```
75          datapath.send_msg(mod)
76
77      def apply_filter_table_rules(self, datapath):
78          parser = datapath.ofproto_parser
79          match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP, ip_proto=in_proto.IPPROTO_TCP)
80          mod = parser.OFPFlowMod(datapath=datapath, table_id=FiREWIRE_TABLE,
81                                  priority=10000, match=match)
82          sh ovs-ofctl -O Openflow13 add-flow s1 table=0,ip,nw_src=10.0.0.16,actions=drop
83          datapath.send_msg(mod)
```

Figura 5.1: *Trecho do código de bloqueio de fluxo por endereço IP, presentes nos cenários vCPE e Sd-VANET*

Com uma única tabela de fluxo, as estruturas executam o código de bloqueio, presente na linha 32 da Figura 5.1, de acordo com a prioridade atribuída a ela. No caso de

uma única tabela, o pacote irá executar a ação de bloqueio e devolverá os demais fluxos para o controlador, que em seguida encaminhará o pacote para as próximas ações, de acordo com cada prioridade. Já no caso das múltiplas tabelas de fluxo, o fluxo originado do *host* 16 será bloqueado, e os demais fluxos serão direcionados para a próxima tabela, que por sua vez, tem suas próprias ações. Neste caso, o pacote não precisa ser devolvido ao controlador, para que ele decida o que fazer com os demais fluxos.

5.2 Cenário vCPE

Para executar o cenário vCPE é preciso definir os parâmetros que devem ser utilizados na estrutura, e quais métricas devem ser utilizadas para avaliar os resultados. Sendo assim, conforme apresentado na Figura 5.3, o cenário vCPE tem 16 *hosts*, definidos com a faixa de *IPs* começando em *10.0.0.1*, e terminando em *10.0.0.16*. A quantidade de *hosts* é definida dentro do código. A Figura 5.2 apresenta o trecho do código que define a quantidade de *hosts*.

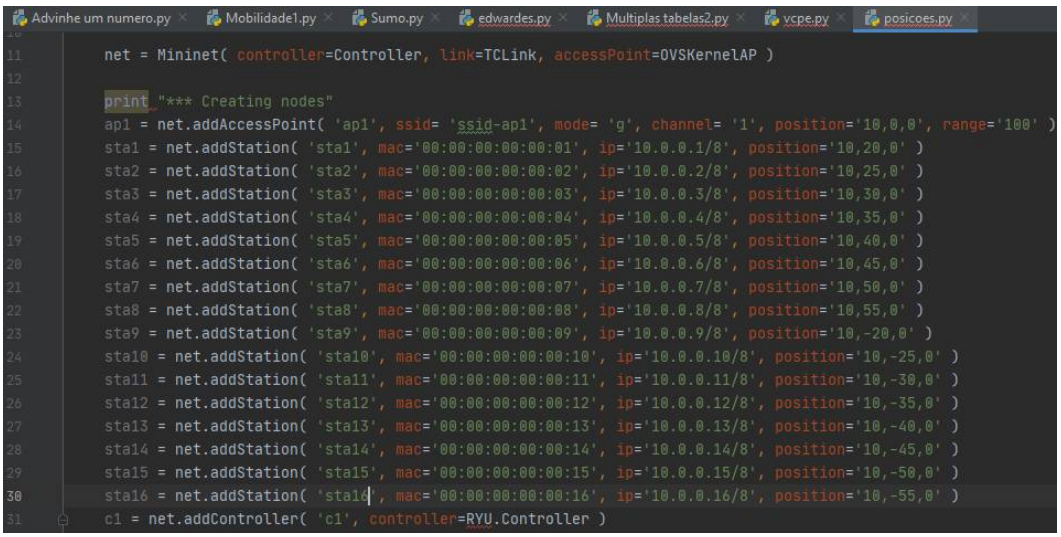

Figura 5.2: *Trecho do código contendo os 16 hosts, presentes nos cenários vCPE e Sd-VANET*

A Figura 5.2 apresenta, da linha 15 até a linha 30, a definição de todos os *hosts* da estrutura, que aqui passam a ser tratados como estações (*sta1, sta2, sta3...,sta16*). Todos os *hosts* devem estar se comunicando entre si. Além disso, de acordo com o comando *position*, todos as estações estão estáticas em relação ao ponto de acesso (*ap1*), que possui como coordenadas os pontos *x=10, y=0 e z=0*. O range de alcance da rede é de 100 metros. A Tabela 5.1 apresenta um resumo dos parâmetros do cenário vCPE.

Tabela 5.1: *Ferramentas, funções e faixa de IPs dos nós que formam o cenário vCPE*

Parâmetros	VirtualBox
Emulador/VM	Mininet-WiFi
Controlador	RYU
Funções	Encaminhamento, Firewall, DHCP e QoS
hosts	16
hosts bloqueados	01
host servidor	01
Faixa de IPs	10.0.0.1/10.0.0.16
Distância Estação/Servidor	55m, no máximo
Software gerador de fluxo	iperf

No emulador Mininet-WiFi, é possível visualizar uma representação gráfica dos nós da rede, por meio do editor *GUI* simples para Mininet-WiFi, chamado *MiniEdit*. Com ele também é possível construir uma rede, configurar elementos da rede, salvar a topologia e executar a simulação. A Figura 5.3 apresenta o gráfico após a criação da topologia da rede.

É importante destacar que as estações representam os mais diversos tipos de equipamentos conectados dos usuários ou clientes. Na topologia apresentada, os nós *wireless* se conectam ao *access point* (*ap1*), que por sua vez, recebe as funções diretamente do controlador RYU.

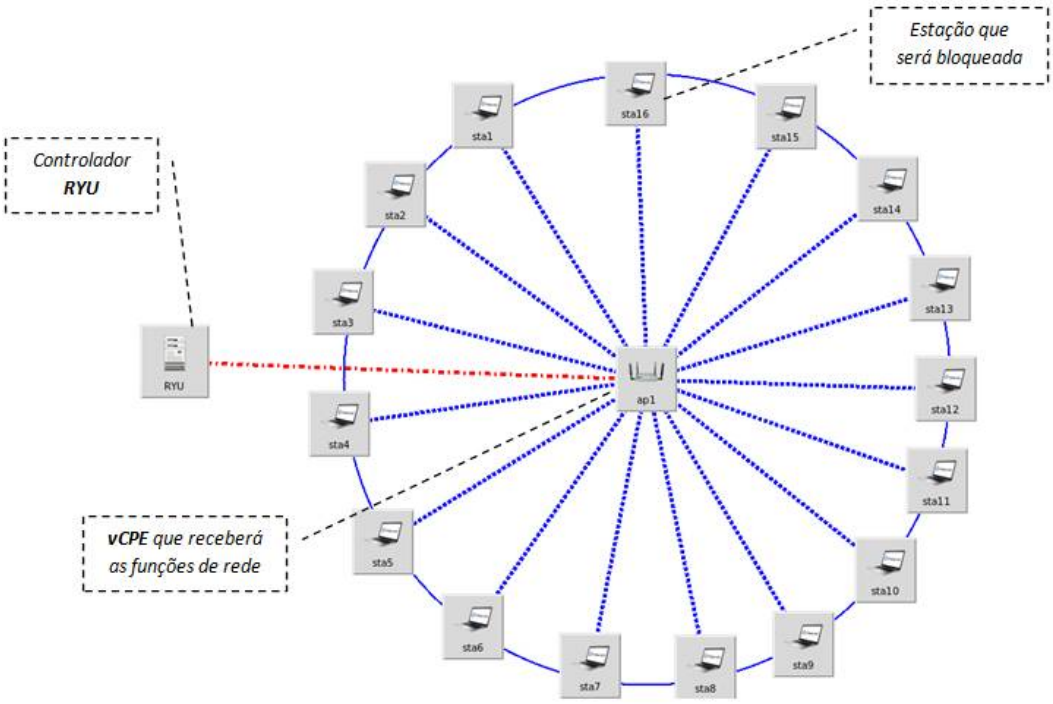

Figura 5.3: *Topologia vCPE instanciada com a função MiniEdit do Mininet-WiFi*

Na Figura 5.3, encontramos a presença dos 16 *hosts* (*sta1, sta2,...,sta16*) conectados com o *access point* (*ap1*), que neste cenário, recebe do controlador RYU, as funções de rede (*Firewall*, Encaminhamento, *DHCP e QoS*). A linha pontilhada azul indica que os *hosts* estãos se comunicando entre si através do *ap1*. A linha pontilhada vermelha indica que o *ap1* está se comunicando e recebendo instruções do controlador RYU. Outra maneira de visualizar a rede em funcionamento, é através da ferramenta, também presente no Mininet-WiFi, chamada de Mininet-WiFi *Graph*. Com ela é possível obter uma visão espectral de toda a rede. A Figura 5.4 apresenta esse gráfico.

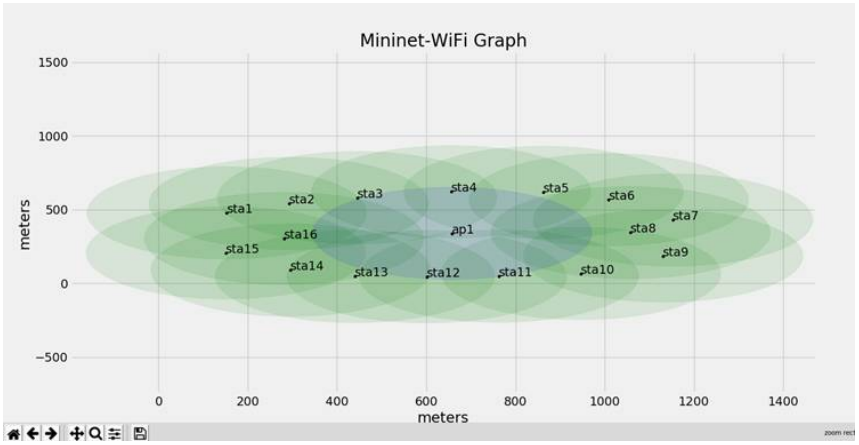

Figura 5.4: *Gráfico da estrutura com 16 estações e o switch instanciada por meio da ferramenta Mininet-WiFi Graph.*

Portanto, o Mininet-WiFi está emulado no *VirtualBox*, sendo executado no sistema operacional *Windows 10*. E a comunicação do plano de dados com o plano de controle é feita pelo RYU. São inseridas quatro funções de rede no cenário vCPE, as funções *Firewall*, Encaminhamento, *DHCP e QoS*. Elas são as funções comumente utilizadas em um ambiente onde existam equipamentos de configuração de clientes, tais como *Modem*, Roteador e *switch*. Cada uma das funções é representada por uma regra ou comando, presente dentro de uma tabela de fluxo individual. A tabela 0 tem a regra de bloqueio de pacote, representando o *Firewall*. A Tabela 1 possui uma regra de Encaminhamento, que direciona imediatamente o pacote para a próxima tabela. A Tabela 2 está configurada com uma função capaz de ativar um servidor *DHCP*, que poderá ser utilizada a qualquer momento, bastando para isso alterar sua prioridade. A Tabela 3 possui regras capazes de dividir a largura de banda da conexão e definir uma quantidade específica para cada conexão, o que equivale a função *QoS*. E por último, na Tabela 4, estão presentes diversos outros campos de correspondência, que podem ser utilizados no caso em que, por ventura, o pacote não tenha encontrado correspondência durante seu percurso. O que, no caso do esquema de múltipla tabela, isso não ocorre. No cenário de

única tabela, as mesmas regras são aplicadas na estrutura, com a diferença que o pacote executa um maior número de saltos. O que acaba prejudicando o desempenho geral da rede.

5.2.1 Execução do Experimento

Após se definir os parâmetros da rede, passamos para a execução do código, contendo a estrutura de 16 *hosts* e 01 *access point*. Primeiro foi executado o experimento na estrutura UTF. Em seguida, para se realizar uma comparação de desempenho, os mesmos testes, contendo os mesmos parâmetros, foram executados na estrutura MTF.

A fim de simular uma comunicação simultânea, entre cliente e servidor, o *iperf* é utilizado para definir que o *host* 01 (*sta1*), execute a função de servidor, recebendo os pacotes de todos os outros *hosts* da rede, também definidos pelo *iperf*. A primeira rodada de testes teve como parâmetros iniciais a taxa de transmissão em 10 *kilobytes* por segundo (*kbps*), transmitindo 01 (um) fluxo, por um período de 60 (sessenta) segundos. Em seguida, foi realizado os mesmos testes, com o aumento gradativo da taxa para 50, 100, 150, 200, 250, 300, 350, 400 e 450. Após a execução de cada ciclo de conexões, os resultados foram coletados e armazenados por *host*. De maneira que do *host2* (*sta2*) até o *host15* (*sta15*), todas as informações de *jitter*, vazão e perdas de pacotes, foram coletadas.

A segunda rodada manteve o mesmo aumento gradativo da taxa de transferência, e aumentou a quantidade de fluxos enviados para o servidor de 01 para 02. De modo que, igualmente, todas as informações de *jitter*, vazão e perdas de pacotes, foram coletadas. Depois de se coletar os resultados utilizando 02 fluxos, aumentamos a quantidade de fluxos para 10 e 20. Cada rodada de testes, mesmo tendo sido definido o tempo de 60 segundos para execução, demorava de 30 a 40 minutos para concluir. Sem dúvida, o equipamento em que os experimentos foram executados influenciou diretamente na capacidade de execução dos testes. Quando se tentou executar a mesma rodada de testes, com mais de 20 fluxos por *host*, o equipamento não suportava a simulação. A Tabela 5.2 apresenta um resumo com todos os parâmetros utilizados para coleta dos resultados.

Com a rede instanciada, via linha de comando, é possível visualizar as estações criadas, o controlador em execução e a comunicação entre os nós da rede. A Figura 5.5 apresenta toda essa dinâmica.

Tabela 5.2: *Resumo dos Parâmetros utilizados nas duas estruturas*

	Única Tabela	**Múltipla Tabela**
Quantidade de *hosts*	16	16
Variação da taxa (KB/s)	10, 50, 100, 150, 200, 250, 300, 350, 400 e 450	10, 50, 100, 150, 200, 250, 300, 350, 400 e 450
Nº de pacotes enviados simultaneamente (por host)	1, 2, 10 e 20	1, 2, 10 e 20
Tempo de envio (seg.)	60	60
Tempo médio aferido (min)	40	38
Métricas avaliadas	*jitter*, *throughput* e perdas de pacotes	*jitter*, *throughput* e perdas de pacotes

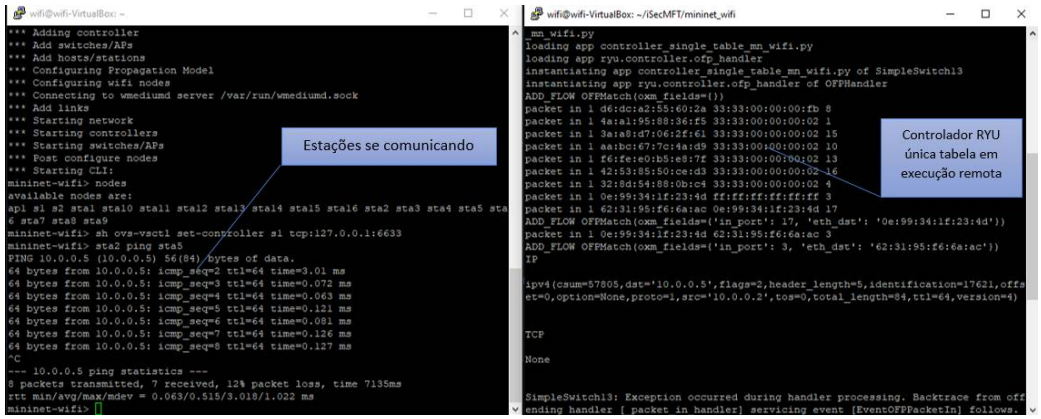

Figura 5.5: *Estrutura com a rede criada. As estações, switches e acces points de um lado. E do outro lado o controlador de única tabela em execução.*

De acordo com a Figura 5.5 a rede é apresentada com um *access point* (*ap1*) e as 16 estações (*sta1, sta2, ...,sta16*). Para testar a comunicação entre os nós, foi utilizado o comando *ping*. Como o Mininet-WiFi é um emulador, e em apenas um computador é possível emular diversos nós ao mesmo tempo, ao realizar uma comunicação, através do *ping* por exemplo, é sempre necessário definir o nó de origem, responsável por esse comando. Por esse motivo que, conforme apresentado na Figura 5.5, o comando *ping* tem o formato *sta2 ping sta5*. Ou seja, é preciso definir que uma determinada estação está querendo se comunicar com outra. Logo após realizar o comando *ping* entre as estações 2 e 5 (*sta2 e sta5*) a comunicação é estabelecida e o tempo de resposta é apresentado. No plano de controle, onde fica o controlador RYU, podem ser vistos atividade de entrada de pacote, o que indica que o controlador está em atividade. No cenário vCPE, a estação sta1 é definida como o servidor que receberá os pacotes gerados pelo *iperf* de todas as 15 outras estações remanescentes. O objetivo é gerar fluxo nas 02 estruturas propostas e verificar seu comportamento.

Conforme definido anteriormente, a primeira rodada de testes é executada na estrutura de única tabela (UTF). Para analisar a quantidade de dados que trafegam por essa estrutura no instante em que todos os 15 nós estão enviando tráfego para a estação servidor (*sta1*), é utilizado o aplicativo *iperf* [31]. Além de avaliar a quantidade de dados que a rede consegue transportar (vazão), também são verificadas as métricas de *jitter* e perdas de pacotes. Depois os testes são executados na estrutura de Múltipla Tabela de Fluxo (MTF).

Como nosso interesse está em verificar o desempenho da estrutura, foi definido o protocolo *UDP* (*User Datagram Protocol*). A taxa de transmissão em UDP pode ser definida pelo usuário, que tem como opção escolher as saídas do resultado entre *KB/s* (*KBytes por Segundo*) ou *MB/s* (*MBytes por Segundo*), para tanto basta acrescentar ao comando o parâmetro *M* para *MB/s* ou *K* para *KB/s* de acordo com a escolha. Para os testes, em cada *host*, essa taxa tem uma variação definida entre 10 a 450 *KB/s*.

Além da variação da taxa de transmissão, foi definido que seria enviada uma quantidade específica de fluxos por um período de tempo também pré-fixado. Então na primeira rodada de testes teríamos em cada *host* o envio de 01 (um) pacote, com a taxa variando entre 10 e 450*KB/s* por um período de 60 (sessenta) segundos. Na segunda rodada de testes teríamos em cada *host* o envio de 02 (dois) pacotes, com a taxa variando entre 10 e 450*KB/s* por um período de 60 (sessenta) segundos. Na terceira rodada de testes teríamos em cada host o envio de 10 (dez) pacotes, com a taxa variando entre 10 e 450*KB/s* por um período de 60 (sessenta) segundos. Na quarta e última rodada de testes teríamos em cada *host* o envio de 20 (vinte) pacotes, com a taxa variando entre 10 e 450*KB/s* por um período de 60 (sessenta) segundos. E assim, dessa forma obtivemos as medidas do *jitter*, vazão (*throughput*) e perdas de pacotes para cada um dos 15 *hosts* da rede.

Após registrar os resultados dos 15 hosts para cada uma das três métricas avaliadas, foi possível obter um valor médio dos resultados, e com isso, conseguimos avaliar as diferenças existentes entre as estruturas UTF e MTF. Primeiro calculamos o valor médio do jitter para 1, 2, 10 e 20 fluxos em cada estrutura. Em seguida realizamos o mesmo cálculo para a vazão (*throughput*) e para as perdas de pacotes. O próximo passo foi calcularmos a variação percentual das médias de cada métrica. Para isso utilizamos a seguinte equação:

$$VariacaoPercentual = (VM1/VM2 - 1) \times 100$$

A Variação percentual é igual ao valor da média 1 (*VM1*) dividido pelo valor da média 2 (*VM2*), menos 1 (um) e multiplicado por 100 (cem). Depois de coletar a variação percentual dos fluxos para as métricas de jitter, vazão e perdas, utilizando UTF e MTF, calculamos a variação média dos percentuais de cada métrica, e finalmente

com o resultado, descobrimos a diferença entre as estruturas UTF e MTF. Em todos os resultados das métricas a seguir, apresentamos uma tabela com a variação do percentual de cada métrica. A Tabela 5.2 apresenta um resumo das métricas e parâmetros dos testes executados.

5.2.2 *Resultados - jitter*

Para a primeira rodada de testes a métrica avaliada foi o *jitter*. O *jitter* é uma variação estatística do atraso na entrega de dados em uma rede, ou seja, pode ser definida como a medida de variação do atraso entre os pacotes sucessivos de dados. Isso implica que uma variação de atraso elevada produz uma recepção irregular dos pacotes. Grandes quantidades de *jitter* indicam baixo desempenho da rede e atraso na entrega do pacote. Quando há alta instabilidade, os pacotes chegam fora da sequencia e se tornam inutilizáveis. Por exemplo, em um sistema telefônico *VoIP*, uma grande latência pode tornar as chamadas indecifráveis [1].

Poucos serviços dependem tanto da entrega de pacotes em tempo real quanto dos sistemas de telefone *VoIP*. Esses sistemas exigem que os pacotes sejam entregues em sequência para serem compreendidos pelo usuário final. Quando os pacotes são transferidos normalmente, eles são enviados da origem ao destino como um fluxo. Numa estrutura vCPE, serviços de telefonia são utilizados com frequência.

Os testes foram executados inicialmente com a estrutura equipada com única Tabela de Fluxo (UTF) e depois com a estrutura equipada com múltiplas tabelas (MTF). Foram coletados os resultados de cada um dos *hosts* presentes nas duas estruturas, e calculado o valor médio entre eles. As Tabelas 5.3 e 5.4 apresentam os resultados e os valores observados com a variação da taxa e do número de pacotes simultâneos, que em cada *host* foram coletados, calculados, e são ilustrados por meio dos gráficos sucessivos às tabelas.

Tabela 5.3: Jitter com 01 Fluxo

	UTF	MTF
Valor médio	0,49	0,41
Variação Percentual	17,36%	

Tabela 5.4: Jitter com 02 Fluxos

	UTF	MTF
Valor médio	0,21	0,18
Variação Percentual	20,77%	

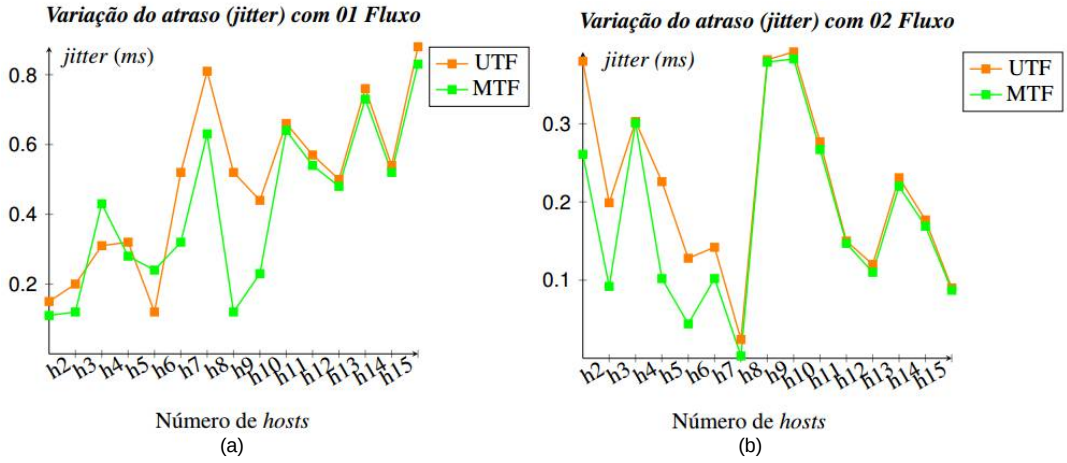

Figura 5.6: *jitter* com 01 e 02 Fluxos simultâneos

As Tabelas 5.3 e 5.4 apresentam o *jitter* com a variação percentual de 17,35% para UTF e 20,77% para MTF. No gráfico da Figura 5.6 são apresentados os resultados obtidos após a realização do experimento, e indica que o fluxo do pacote com a estrutura de múltipla tabela tem um valor de *jitter* menor em relação aos valores da estrutura UTF. Nas duas estruturas, se percebe que para 01 (um) fluxo os números começam baixos, em torno de 0,2 (*ms*) e atingem picos de quase 1 (*ms*). Essa variação ocorre, pois conforme definido anteriormente, os nós da rede estão em posições estáticas, mas com distâncias diferentes. De acordo com as coordenadas definidas na Seção 5.2, os primeiras e as últimas estações estão cada vez mais próximas da estação servidor.

Agora com o envio de 02 fluxos simultâneos, a Figura 5.6 (b) apresenta os resultados obtidos após a realização do experimento, e indica que o fluxo do pacote com a estrutura de múltipla tabela também tem um valor de *jitter* menor em relação aos valores da estrutura UTF. Diferentemente do experimento realizado com 01 fluxo apresentado na Figura 5.6 (a), as variações do atraso com dois fluxos simultâneos, começam com valores altos em relação ao restante das conexões, e nos nós mais distantes do ponto de acesso o *jitter* é maior para as duas estruturas, chegando a alcançar quase 0,4*ms*.

Tabela 5.5: Jitter com 10 Fluxos

	UTF	MTF
Valor médio	1,98	1,59
Variação Percentual	24,63%	

Tabela 5.6: Jitter com 20 Fluxos

	UTF	MTF
Valor médio	13,96	12,58
Variação Percentual	10,95%	

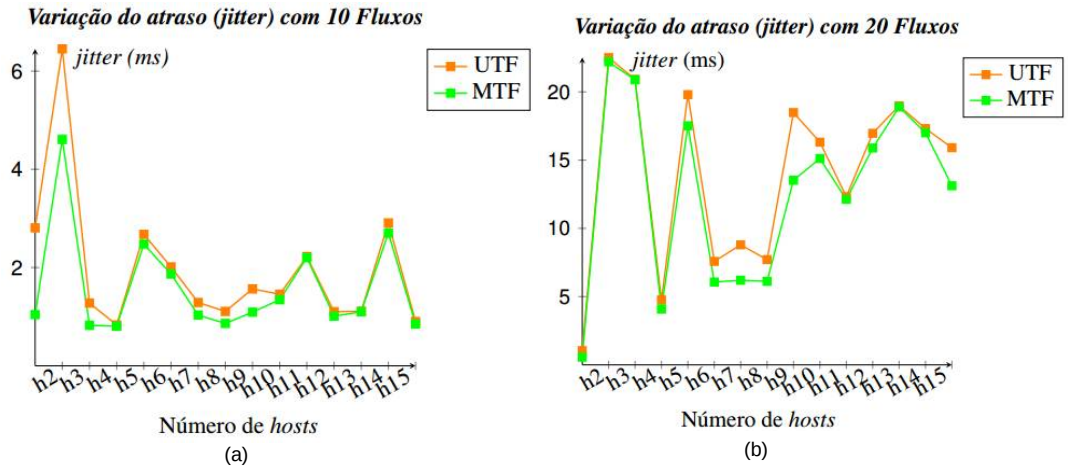

Figura 5.7: *jitter* com 10 e 20 Fluxos simultâneos

As Tabelas 5.5 e 5.6 apresentam a variação percentual de 24,63% e 10,95% para 10 e 20 fluxos. Com 10 conexões simultâneas, tanto para a estrutura UTF quanto pra estrutura MTF, os valores de *jitter* passam de 0,4*ms*, com o *host* 3 da estrutura UTF ultrapassando 0,6 *ms*, o que ainda continua aceitável dentro de uma rede sem fio, que mesmo atingindo 30ms, conseguem entregar seus pacotes, desde que não tenham perdas de pacotes superiores a 1% [1]. A Figura 5.7 apresenta os resultados obtidos após a realização do experimento, e mais uma vez indica que o fluxo do pacote com a estrutura de múltipla tabela tem um valor de *jitter* menor em relação aos valores da estrutura UTF. As unidades de *jitter* na maioria dos *hosts* das duas estruturas permanecem com valores parecidos.

A Figura 5.7 apresenta os resultados obtidos após a realização do experimento, e mais uma vez indica que o fluxo do pacote com a estrutura de múltipla tabela tem um valor de *jitter* menor em relação aos valores da estrutura UTF. As unidades de *jitter* na maioria dos *hosts* das duas estruturas, permanecem com valores bem próximos. Com 20 conexões simultâneas, tanto para a estrutura UTF quanto pra estrutura MTF, os valores de *jitter* chegam a atingir quase 25ms com o *host* 3 das duas estrutura. O que a partir daqui passa a preocupar, pois, se a perda de pacotes for maior que 1%, a rede começa a ter dificuldades na entrega [1].

5.2.3 Resultados - Vazão

Vazão (*throughput*), é a quantidade de dados isentos de erros transferidos com sucesso entre dois nós por unidade de tempo. Uma condição ideal seria a vazão ser igual à capacidade da rede, o que dificilmente ocorre na prática. Para análise da rede, a vazão é

medida em bits por segundo (*bps*), podendo, ainda, ser medida em megabits por segundo (*Mbps*), ou ainda gigabits por segundo (*Gbps*).

Para este experimento foi definido que a rede poderá atingir até 2 megabits por segundo (*Mbps*). O objetivo de se utilizar essa métrica é analisar a quantidade de dados que é transferida pela rede com UTF e compará-la com a rede MTF. Neste cenário, a rede que obtiver valores maiores de vazão, é considerada mais eficiente do que as que apresentam valores menores.

Durante a execução do experimento, foi possível aferir a taxa em que os dados são transmitidos para os cenários equipados com UTF e MTF. A aferição do *throughput* foi inicialmente realizada aplicando 01 fluxo. Em seguida, foram executados testes com 2, 10 e 20 fluxos. Na Figura 5.8 (a) apresenta-se o gráfico para 01 fluxo, e na Figura 5.8 (b) apresenta-se o gráfico para 02 fluxos. De acordo com a Figura 5.8, o *throughput* para a rede equipada com MTF é maior, ou seja, chega mais próximo de conseguir transmitir dados com uma taxa de 2 (*Mbps*). No gráfico, os nós representados pela linha de cor verde se sobrepõem durante todo o percurso aos nós representados pela linha de cor laranja.

Tabela 5.7: *Throughput* com 01 Fluxo

	UTF	MTF
Valor médio	1823,1	1859,1
Variação Percentual	1,97%	

Tabela 5.8: *Throughput* com 02 Fluxos

	UTF	MTF
Valor médio	1741,8	1760,3
Variação Percentual	1,06%	

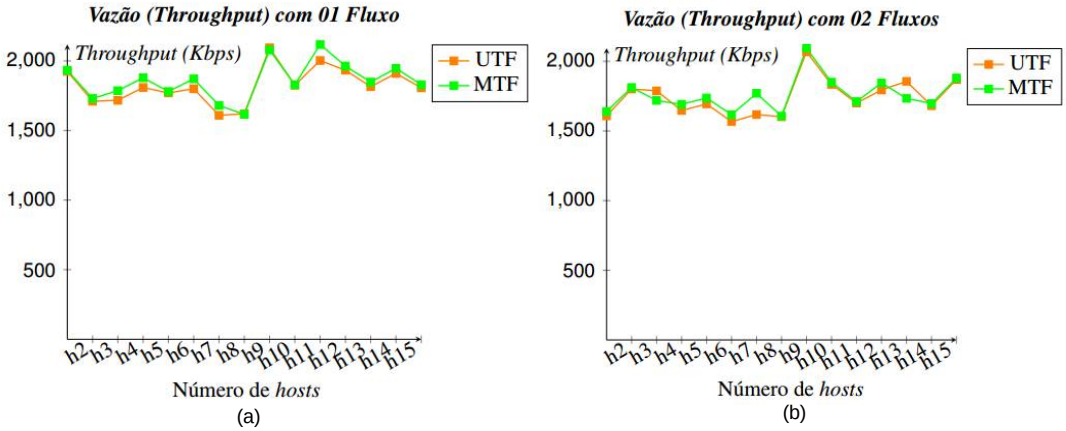

Figura 5.8: *Vazão com 01 e 02 Fluxos simultâneos*

As Tabelas 5.7 e 5.8 apresentam a variação percentual do *throughput* para 1 e 2 fluxos, 1,97% e 1,06% respectivamente. Com 1 fluxo simultâneo, a vazão se

mantêm próximo aos 2 *Mbps* previamente estabelecidos. Apresentando inclusive, um comportamento equivalente nos *hosts* 10 e 12 equipados com MTF. Com 2 fluxos simultâneos, conforme podemos observar no gráfico da Figura 5.8 (b), o *throughput* sofre uma leve queda e passa a assumir uma taxa superior a 1,5 (*Mbps*), atingindo em um determinado momento a taxa de 2 *Mbps* (nó 10). E para a rede equipada com MTF, conforme indicado pelos nós representados no gráfico pela linha de cor verde, esse valor é maior. Conforme pode ser constatado durante quase todo o percurso (com exceção do nó 4 e do nó 13). Isso, mais uma vez indica que a estrutura equipada com MTF, continua a manter um desempenho superior à estrutura equipada com UTF.

Tabela 5.9: *Throughput* com 10 Fluxos

	UTF	MTF
Valor médio	1067,5	1074,9
Variação Percentual	0,69%	

Tabela 5.10: *Throughput* com 20 Fluxos

	UTF	MTF
Valor médio	745,9	755,4
Variação Percentual	1,26%	

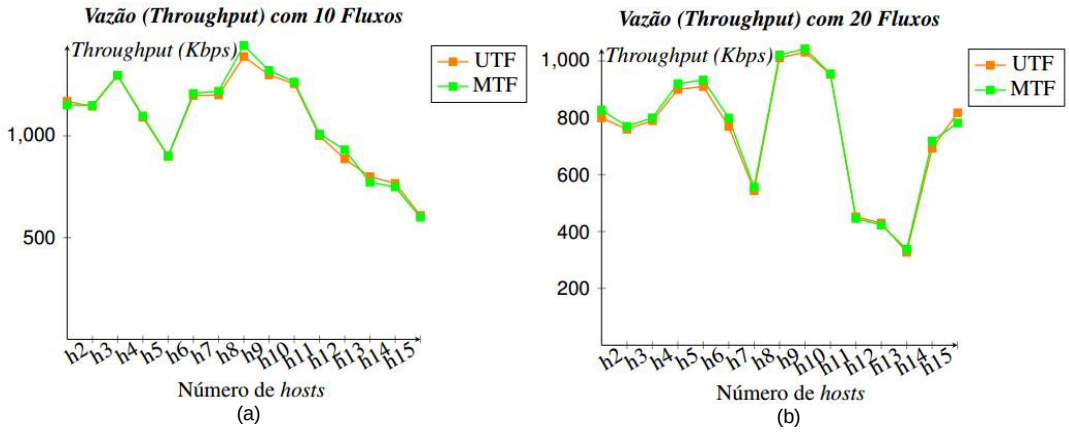

Figura 5.9: *Vazão com 10 e 20 Fluxos simultâneos*

As Tabelas 5.9 e 5.10 apresentam a variação percentual do *throughput* para 10 e 20 fluxos, e os valores se mantém bem próximos para as duas estruturas, 0,69% e 1,26%. Com 10 fluxos simultâneos, as estruturas UTF e MTF sofrem uma importante diminuição do seu *throughput*. Conforme apresentado na Figura 5.9, os valores caem de 1,2 (*Mbps*), para um pouco mais de 500 *Kbps*. Mesmo assim, se obtém uma taxa superior utilizando a estrutura MTF. É possível observar nos gráficos das Figuras 5.9 (a) e 5.9 (b), que a linha verde, que representa a estrutura MTF, se sobrepõe a linha de cor laranja, que

representa a estrutura UTF. Enviando 20 fluxos simultaneamente, a rede consegue manter uma estabilidade em relação ao experimento utilizando 10 fluxos simultâneos. De acordo com a Figura 5.9, a taxa para os *hosts* 2, 3, 4, 5, 6, 9 e 10 das duas estruturas chegam a ficar entre 0,8 a 1 (*Mbps*). Nos *hosts* 7, 8, 12 a 14 a taxa fica entre 0,3 e 0,8 (*Mbps*). Com isso é possível verificar uma maior oscilação do *Throughput* quando se aplica 20 fluxos simultâneos.

5.2.4 Resultados - Perdas de pacotes

Os usuários que utilizam a Internet ou acessam outras redes online estão sujeitos perder suas informações através do envio e recebimento de unidades de dados que são referidos como pacotes. Durante todo esse processo, vários pacotes podem ser perdidos na transferência e não conseguir chegar ao endereço de destino. Isto é referido como perda de pacotes. Ele pode ser experimentado na forma de serviços lentos, interrupções na conexão de rede e perda total de conectividade de rede. A perda de pacotes pode afetar qualquer aplicativo, mas é mais provável que interrompa aqueles que dependem de transferências de dados em tempo real, como programas que transmitem áudio ou vídeo, juntamente com outras várias plataformas de sistemas digitais [2]. E estes serviços, geralmente devem estar disponíveis nos vCPEs. A partir da Figura 5.10, é possível analisar os pacotes de dados perdidos, durante os experimentos utilizando as estruturas UTF e MTF.

Tabela 5.11: Perdas de pacotes, 01 Fluxo

	UTF	MTF
Valor médio	0,29	0,22
Variação Percentual	30,63%	

Tabela 5.12: Perdas de pacotes, 02 Fluxos

	UTF	MTF
Valor médio	0,34	0,22
Variação Percentual	53,25%	

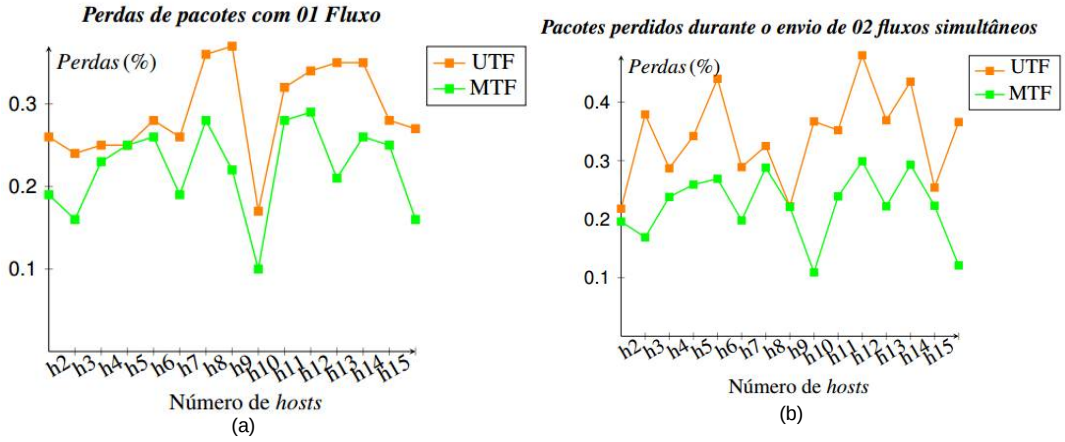

Figura 5.10: *Perdas de pacotes com 01 e 02 Fluxos simultâneos*

As Tabelas 5.11 e 5.12 apresentam uma variação percentual de 30,63% e 53,25% para o total de pacotes perdidos nas estruturas UTF e MTF. Com o envio de 01 fluxo simultâneo, é possível notar no gráfico da Figura 5.10 (a), que durante a comunicação com o servidor, a rede sofre uma pequena perda de pacotes. Nos *hosts* 2 ao 6, essa perda fica entre 0,1 e 0,3 % na estrutura equipada com MTF. Na estrutura equipada com UTF, essa perda, nos mesmos *hosts*, é maior, ficando entre *0,2% e 0,3%*. Os *hosts* 7 e 8 da estrutura UTF sofre uma perda maior em relação a estrutura MTF, chegando a ultrapassar os 0,3% do total de perdas. E nos *hosts* 10 a 15 se mantém próximos de *0,3%* de perdas. De acordo com o gráfico da Figura 5.10 (b), os fluxos dos nós 2 ao 15 da estrutura UTF, sofrem a perda de *0,2% a 0,5%* do total de seus pacotes. Já na estrutura MTF, nos mesmos nós, essa perda é menor, ficando entre *0,1% e 0,2%* do total de perdas.

Tabela 5.13: Perdas de pacotes, 10 Fluxos

	UTF	MTF
Valor médio	1,82	1,63
Variação Percentual	12,05%	

Tabela 5.14: Perdas de pacotes, 20 Fluxos

	UTF	MTF
Valor médio	6,8	6,5
Variação Percentual	3,9%	

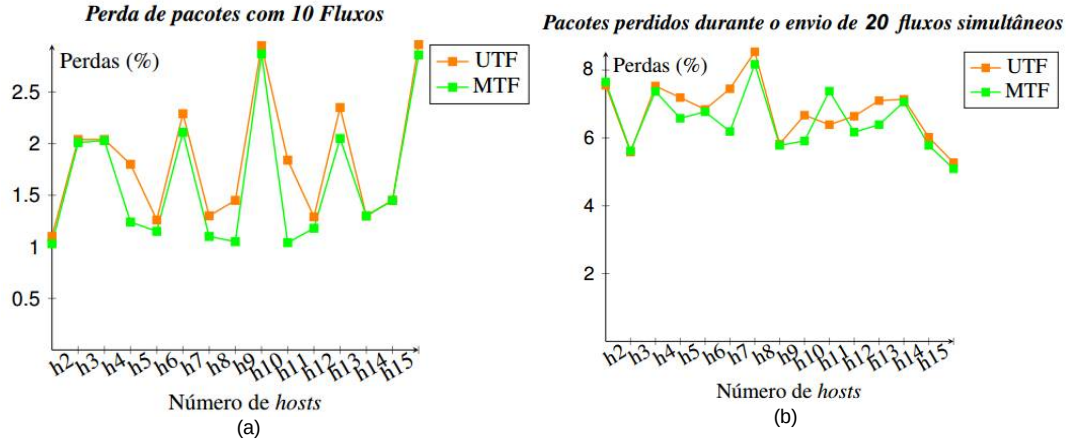

Figura 5.11: *Perdas de pacotes com 10 e 20 Fluxos simultâneos*

As Tabelas 5.13 e 5.14 apresentam os resultados do valor médio das perdas dos pacotes nas estruturas UTF e MTF, obtendo uma variação percentual de 12,05% e 3,90% respectivamente. De acordo com o gráfico presente na Figura 5.11, com todos os 15 nós enviando pacotes simultâneos para o servidor (*sta1*), a rede começa a sofrer perdas significativas. A partir do primeiro nó (*sta2*), já se observa uma perda de *1%* dos pacotes. Na sequência, essa perda vai aumentado e diminuindo, oscilando entre *1% e 2,5%*. Nos *hosts* 9 e 10 as perdas chegam a ultrapassar os *2,5%*, tanto na estrutura UTF quanto na estrutura MFT. No gráfico da Figura 5.11 (b) apresenta-se os resultados das estruturas UTF e MFT durante o envio de 20 fluxos simultâneos. As perdas de pacotes ficam em média entre *6% e 8%* para as duas estruturas.

5.2.5 Discussão dos Resultados - vCPE

Para aferir o desempenho das estruturas vCPEs equipadas com UTF e MTF, foram feitos testes, pelos quais se pode confrontar o desempenho de uma rede composta de 16 *hosts* enviando ao mesmo tempo, diferentes quantidades de pacotes, com uma taxa variando entre 10 e 450 *KB/s*, por um período pré definido de 60 segundos. Para avaliar o experimento, foram utilizados as métricas de *jiiter*, *throughput* e perdas de pacotes. Cada métrica utilizando 01, 02, 10 e 20 fluxos simultâneos. Os resultados são apresentados nos gráficos plotados nas Figuras 5.6 até a Figura 5.11.

A primeira métrica avaliada foi o *jitter*. A quantidade de *jitter* de rede aceitável em uma rede (muitas vezes referida como *jitter* aceitável) depende do tipo de serviço que você está usando. Alguns aplicativos e serviços têm um nível de tolerância mais alto para *jitter* do que outros. Por exemplo, o *jitter* não afeta o envio de *e-mails* tanto quanto afetaria um *chat* de voz.

Ao usar aplicativos como soluções de VoIP com baixa tolerância para *jitter*, deve-se garantir que o *jitter* seja mantido abaixo de 30 milissegundos. Qualquer taxa de *jitter* abaixo deste valor será aceitável porque os efeitos do *jitter* serão mínimos. *Jitter* mínimo significa que os usuários ainda serão capazes de compreender a pessoa do outro lado da chamada, por exemplo. Um vCPE que esteja oferecendo *VoIP*, *streaming* de vídeo, e outros serviços que precisam ser prestados com segurança e controle, que é exatamente o que as funções de *Firewall*, Encaminhamento, *DHCP* e *QoS* propõem, devem ter um nível de *jitter* aceitável durante um determinado processo.

Conforme apresentado no gráfico da Figura 5.6, utilizando 01 (um) fluxo, o experimento apresentou uma variação de atraso na entrega de pacotes muito baixa, 0,2 (*ms*) em média. Isso não produz efeitos negativos na estrutura. A mesma situação se repete no gráfico do lado na Figura 5.6 quando se utiliza 02 fluxos simultâneos. Com 10 fluxos simultâneos, o gráfico da Figura 5.7 revela que o *jitter* da estrutura UTF atinge um pico de 6 (*ms*), mas na sequencia permanece com um valor médio baixo, por volta de 2 (*ms*), nas duas estruturas. A situação para 20 fluxos simultâneos sofre um importante aumento, de acordo com o gráfico da Figura 5.7 (b), a rede chega a atingir 20 (*ms*) de *jitter*. Contudo, o ponto importante a ser observado na variação do atraso na entrega dos pacotes, é que, em todos os casos apresentados, a estrutura equipada com o esquema MTF, manteve um valor de *jitter* inferior ao da estrutura UTF. Essa situação revela que o cenário vCPE que utiliza o esquema flexível de múltiplas tabelas para acoplar suas funções, também apresenta um desempenho total melhor. A Tabela 5.15 revela que a estrutura MTF teve uma variação média de atraso 18,43% menor que a estrutura UTF.

Tabela 5.15: *Variação Percentual Média para o jitter*

	UTF e MTF			
	01 Fluxo	02 Fluxos	10 Fluxos	20 Fluxos
Variação Percentual (%)	17,36	20,77	24,63	10,95
Variação Percentual Média	18,43%			

O cálculo para encontrar a variação percentual média para a variação do atraso descrito na Tabela 5.15 é bem simples. Para isso, basta somar as médias das variações do percentual do valor médio de cada host e dividir pela sua quantidade. Dessa forma temos:

$$\frac{vpm = \sum vp}{4}$$

Assim, o Valor Percentual Médio (*vpm*) é igual ao somatório do Valor Percentual (*vp*), dividido por quatro (*4*). E este mesmo cálculo é repetido para as métricas *throughput*

e perdas de pacotes.

Ao avaliar o *throughput* para as estruturas UTF e MTF é possível verificar que, utilizando apenas 01 (um) fluxo simultâneo, a rede atinge a taxa de 2.000 (*Kbps*) (ou 2 *Mbps*). Na Figura 5.8 é possível constatar que a estrutura equipada com MTF consegue manter uma taxa superior, e portanto melhor que a estrutura UTF. Conforme apresentado no gráfico do lado direito da Figura 5.8, com 02 (dois) fluxos a taxa reduziu para 1,5 (Mbps) em média para as duas estruturas. Contudo, a estrutura MTF continua mantendo um comportamento superior. O gráfico da Figura 5.9 com o envio simultâneo de 10 fluxos, apresenta uma variação que fica entre 500 *Kbps* e 1500 *Kbps*, o que é justificado pelo o aumento de fluxos simultâneos. No gráfico do da Figura 5.9 (b), a vazão sofre uma importante queda, saindo de 800 *Kbps* e chegando a pouco mais de 300 *Kbps*. O que já é esperado, tendo em vista que o aumento dos fluxos simultâneos produz diminuição no *throughput* geral da rede. No entanto, a estrutura MTF mantém um valor médio melhor que a estrutura UTF. A Tabela 5.16 apresenta os resultados da Variação Percentual Média (*vpm*) para o *throughput*.

Tabela 5.16: *Variação Percentual Média para o throughput*

	UTF e MTF			
	01 Fluxo	02 Fluxos	10 Fluxos	20 Fluxos
Variação Percentual (%)	1,97	1,06	0,69	1,26
Variação Percentual Média	1,25%			

Outra métrica avaliada foi a perda de pacotes durante a transmissão de diferentes quantidades de fluxos. Perder pacotes de dados é uma situação que pode causar impactos negativos no funcionamento de toda a rede, pois as informações que deveriam chegar corretamente a um determinado destino podem ser corrompidas. Nos experimentos utilizando o cenário vCPE, a rede começou a apresentar importantes perdas a partir da transmissão de 20 fluxos, aplicados de maneira simultânea através dos 15 *hosts*. Com a aplicação de apenas 01 fluxo simultâneo, as duas estruturas não chegaram a perder *1%* dos pacotes, se mantendo entre *0,2% e 0,3%*. A situação se repetiu aplicando-se 02 fluxos simultâneos. Com 10 fluxos simultâneos as duas estruturas chegaram a perder até *3%* dos seus pacotes enviados. Todos os pacotes perdidos, durante a transmissão das diferentes quantidades de fluxos simultâneos, desferidos por todos os 15 nós das estruturas UTF e MTF, não causam impactos significativos na maioria das aplicações. A não ser que a perda de pacotes seja superior a *30%* do total, o que não ocorreu nos experimentos apresentados. Nesses casos a variação do atraso na entrega, por não ser tão alto, gerou uma influencia positiva em toda a conexão. A Tabela 5.17 apresenta os resultados da Variação Percentual Média (*vpm*) para os pacotes perdidos nas estruturas UTF e MTF.

Tabela 5.17: *Variação Percentual Média para as perdas de pacotes*

	UTF e MTF			
	01 Fluxo	02 Fluxos	10 Fluxos	20 Fluxos
Variação Percentual (%)	30,63	53,25	12,05	3,90
Variação Percentual Média	24,96%			

Contudo, o ponto mais importante a ser considerado nos experimentos, é o fato de todos os resultados terem apresentado valores positivos para a estrutura equipada com MTF. Isso é significativo, pois reforça o que está sendo proposto neste trabalho. Portanto, estruturas equipadas com MTF tem a capacidade de tornar o cenário vCPE mais flexível e com um desempenho superior ao da estrutura equipada com UTF. Portanto, para ampliar os limites de usabilidade eficiente das estruturas equipadas com MTF, a seção 5.3 apresenta o cenário de VANETs definidas por software, equipadas com MTF. E realiza a comparação do mesmo cenário equipado com a estrutura UTF.

5.3 Cenário - Sd-VANET

Em geral as Sd-VANETs tem o objetivo de fornecer gerenciamento inteligente e eficaz de transito (ITS). Sendo assim, a ideia de programar redes com *switches* equipados com múltiplas tabelas de fluxo também pode ser aplicada nas arquiteturas Sd-VANETs [10].

Nessa Seção, apresentamos uma maneira eficiente de implementar os RSUs presentes nas Sd-VANETS. Em um ambiente onde essas redes podem operar no modo infraestruturada, uma RSU pode apropriadamente armazenar o plano de dados da SDN. Ou seja, o *switch OpenFlow* pode ser instanciado dentro dessa unidade. Desta forma, as Sd-VANETs podem ser equipadas com várias funções e serviços. Segue o exemplo de algumas delas:

Prevenção de colisão em um cruzamento: A possibilidade de uma colisão é alta em um cruzamento. Aproximar-se do veículo que esteja mais próximo do cruzamento poderá fazer com que ele envie mensagens de alerta para outros veículos próximos para evitar o risco de uma colisão lateral [33].

Pelotão de veículos: Aumenta a segurança rodoviária para permitir veículos em um pelotão, abolindo o problema da mudança de faixa e controle de velocidade. Assim, o pelotão se beneficia do aumento da economia de combustível e da redução do congestionamento nas estradas e das colisões de tráfego [10].

Luzes de freio eletrônicas: O veículo alerta outros usuários que a desaceleração está ocorrendo na frente deles para evitar colisões traseiras [33].

Tabela 5.18: Classificação das aplicações VANET

Classe de aplicação	Característica a considerar	Arquitetura	Tecnologia de comunicação	Propriedades desejáveis nos protocolos
Segurança	Delay	V2V-V2I	DSRC/RFID/ Bluetooth/WiFi	Confiabilidade
Eficiência	Disponibilidade	V2V-V2I	DSRC	Tempo real e Confiabilidade
Conforto	Disponibilidade	V2I	WiMAX/WiFi/3G/ 4G/LTE	Tempo real
Entretenimento Interativo	Conectividade e disponibilidade	V2V-V2I	WiMAX/WiFi/3G/ LTE	Comunicação *Unicast*
Sensação Urbana	Mobilidade	V2V-V2I	DSRCWiMAX/ WiFi/3G/LTE	Coleta de dados

Aviso de violação de sinal de trânsito: Unidades na estrada detectam violação de sinal de trânsito. Essas informações são transmitidas ao centro de controle de tráfego e outros veículos [10].

Sensor Pré-colisão: O sistema de segurança Pre-colisão inclui tecnologia de radar de ondas milimétricas para ativar os freios pré-colisão e preparar os cintos de segurança pré-colisão para reduzir lesões [33].

Notificação pós-acidente: Veículo envolvido em acidente ou unidade de beira de estrada envia mensagens de advertência a outros veículos a seguir, para que o motorista possa escolher uma rota alternativa. Isso reduz o congestionamento no ponto de acidente [33]. A Tabela 5.18 apresenta uma classificação com as principais aplicações, arquiteturas, tecnologias e propriedades necessárias em uma VANET.

Todas essas funções ficam mais bem alocadas em uma estrutura equipada com múltiplas tabelas de fluxo, pois quando as funções ficam distribuídas e organizadas no esquema MTF, toda a estrutura passa a ganhar flexibilidade e ganho de desempenho [3]. Sendo assim, para apresentar a eficácia das múltiplas tabelas também em Sd-VANETs, realizamos a simulação de uma VANET definida por *software* equipada com múltiplas funções, distribuídas em MTF. A simulação é realizada no Mininet-WiFi. Assim como no cenário vCPE, o cenário Sd-VANET possui 16 *hosts*, tratados no Mininet-WiFi como estações. Na implementação, as estações devem se comunicar umas com as outras através do meio sem fio. Também se comunicam com o *switch OpenFlow*, que na simulação está acoplado a RSU da VANET. Isso significa que o *switch* é uma abstração do RSU da VANET. E essa abstração ocorre em um ambiente que simula o tráfego rodoviário. Para isso se utiliza um aplicativo instalado dentro do emulador Mininet-WiFi chamado de SUMO (*Simulation of Urban MObilty*) [5] [53].

O SUMO é um *software* de simulação de mobilidade urbana de código aberto, que pode ser utilizado na modelação de sistemas intermodais de transporte, incluindo fluxos de tráfego de veículos privados e de transportes públicos. Além de ser capaz de realizar uma simulação microscópica, com modelação explícita de veículos privados, de transportes públicos, o SUMO pode ser utilizado numa multiplicidade de tarefas como a avaliação do desempenho de sistemas de semaforização, no planeamento e gestão de rotas, ou na avaliação do comportamento de redes de comunicação [53]. A Figura 5.12 apresenta o ambiente *GUI* deste aplicativo.

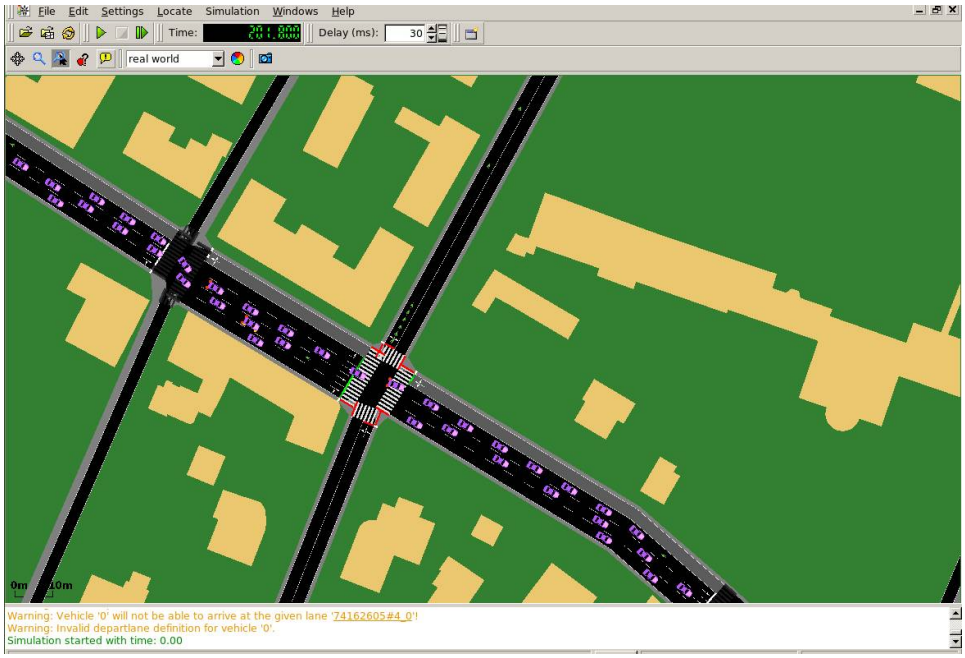

Figura 5.12: *Ambiente GUI do aplicativo SUMO*

A Figura 5.12 apresenta o ambiente do aplicativo SUMO, responsável pela simulação do tráfego rodoviário. Os carros simulados pelo SUMO, são tratados como estações pelo Mininet-WiFi, e no experimento utilizando o cenário Sd-VANET, são chamados de *car* (*car1, car2, car3,..., car16*). As estações são configuradas com endereços *IPs* começando em *10.0.0.1* para a estação 01 (car1), e terminando em 10.0.0.16 para a estação 16 (car16). Como os nós estão sendo avaliados de maneira estática, a única política de mobilidade aplicada é a definição de suas coordenadas geográficas no código de criação. Estas coordenadas seguem o mesmo modelo estabelecido no cenário vCPE, com a diferença de estarem mais dispersados em relação ao nó central. Assim, o *access point* (*ap1*) recebe a coordenada *x=10, y=0 e z=0*, e as estações recebem as coordenadas começando em *x=10, y=20* e *z=0*. A Figura 5.13 exibe os detalhes das coordenadas e parâmetros de todos as 16 estações.

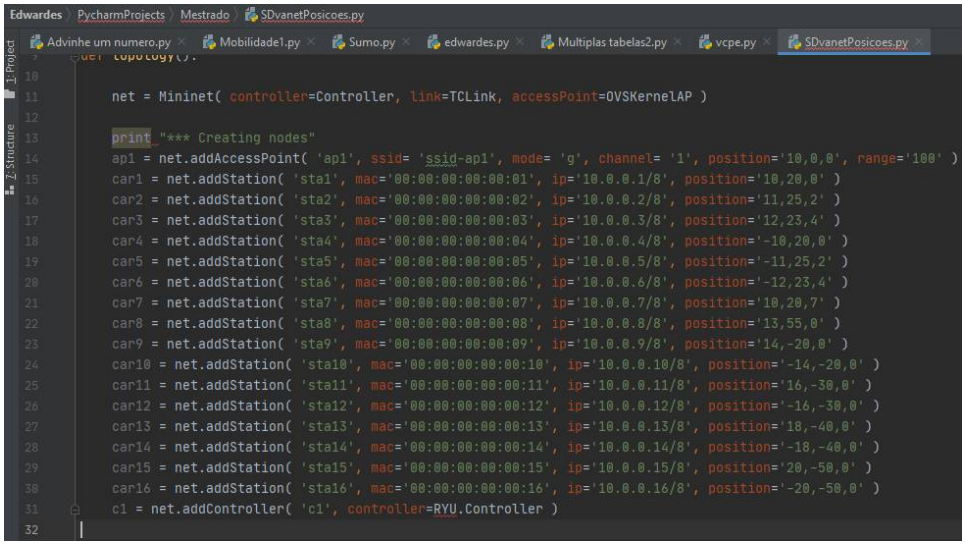

Figura 5.13: T*recho do código com 16 hosts contendo as posições geográficas da Sd-VANET*

Na Figura 5.13, percebemos que as coordenadas das estações (*car1, car2, car3, ..., car16*) possuem posições distintas umas das outras, aumentando a distancia de forma gradual em relação ao nó central (*ap1*). O objetivo é deixar o cenário compatível com uma Sd-VANET. Toda a produção da topologia da rede é realizada via *script* pela linguagem de programação *Python*. Esta linguagem foi utilizada para programar as classes que fizeram a modelagem de toda a topologia, incluindo os nós e demais elementos da rede [54]. Dessa maneira, fluxos estáticos são instalados e modificados no *switch OpenFlow* com o objetivo de emular a mobilidade dos carros de maneira controlada, ou seja, baseada em *script*. Este *switch* recebe do controlador RYU os parâmetros que o define como *switch* equipado com UTF ou MTF.

O *script* responsável pela instalação de fluxos estáticos contém instruções pré-programadas para todas as conexões. Como o Mininet-WiFi fornece suporte à mobilidade veicular, o que inclui o acoplamento com o gerador de padrões de mobilidade SUMO realista, o experimento pode ser melhor definido especificando os modelos de mobilidade de escolha e tendo um aplicativo SDN gerenciando as operações de rede com base na posição dos nós, na qualidade do sinal de rádio e assim por diante [53] [13]. A Figura 5.14 apresenta a topologia Sd-VANET instanciada logo após a definição dos parâmetros e execução do código.

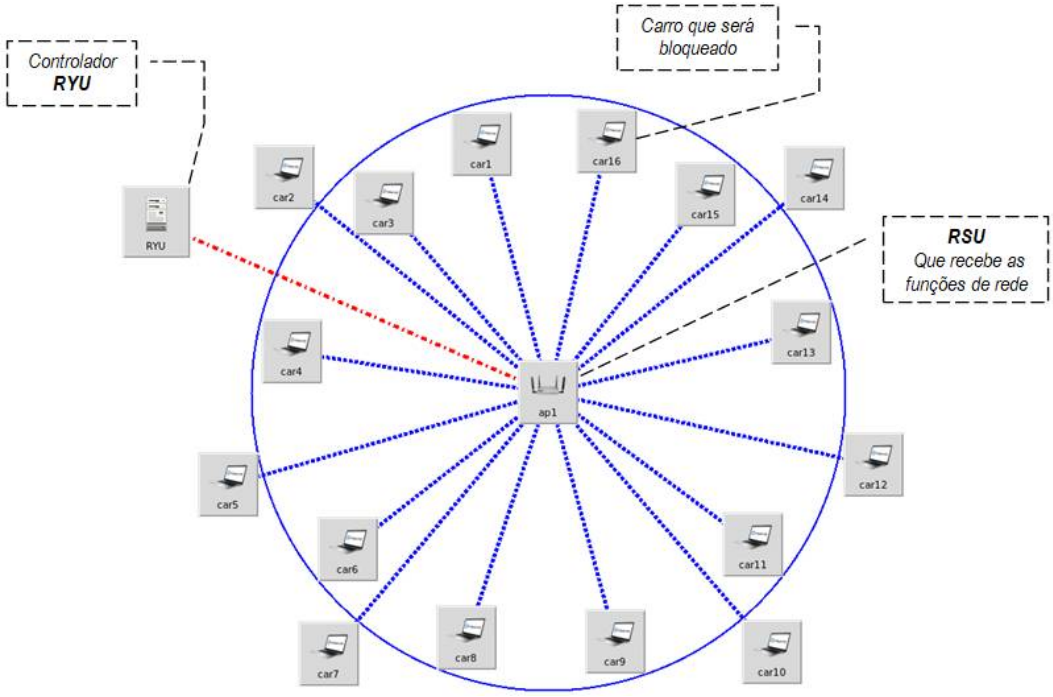

Figura 5.14: *Topologia Sd-VANET instanciada com a função MiniEdit do Mininet-WiFi*

Na Figura 5.14 está presente o *access point* (*ap1*), que nesse experimento representa o RSU da Sd-VANET. Também está presente as 16 estações, que no experimento podem representar os OBUs (*On-Board Units*) e também podem representar os carros da Sd-VANET. A principal diferença em relação ao cenário vCPE está no posicionamento geográfico dos carros em relação ao ponto central (RSU), que apesar de permanecerem estáticos, possuem posições diferentes das dispostas no vCPE, estando portanto mais dispersos em relação ao ponto central. Na Tabela 5.3 apresentamos um resumo com os parâmetros da rede.

5.3.1 Execução do Experimento

No experimento, os carros (estações), estão trafegando dentro do raio de alcance do *ap1*, que está operando no modo *G* e que encontra-se estático. Seu raio de alcance é de 180 *m*. Operando no padrão *802.11g* e com a frequência de 2,4 *GHz*, a rede pode alcançar a velocidade máxima de 54 *Mbits/s*. No experimento Sd-VANET as estações estão configuradas para atingirem até 2 *Mbits/s*, disparando contra o servidor 01 fluxo simultâneo por estação na primeira rodada de execução. Dois fluxos simultâneos na segunda rodada. Dez e vinte fluxos simultâneos na terceira e quarta rodada de testes. Todos as rodadas sendo executadas com as taxas de 10 a 450 *KB/s*. A Tabela 5.3 apresenta os detalhes do cenários. Além disso, na simulação, cada carro é representado

Tabela 5.19: *Resumo dos Parâmetros Utilizados no Cenário Sd-VANET*

Parâmetros/Ferramentas	**Única Tabela**	**Múltipla Tabela**
Emulador	Mininet-WiFi	
Controlador	RYU	
Métricas avaliadas	*Jitter*, *throughput* e perdas	
Variação da taxa (*KB/s*)	10, 50, 100, 150, 200, 250, 300, 350, 400 e 450	
Nº de pacotes enviados simultaneamente	1, 2, 10 e 20	
Tempo de envio (seg.)	60	
Tempo médio aferido (min)	47	45
App gerador de fluxo	*iperf*	

por uma estação móvel, equipada com uma placa de rede sem fio, operando na frequência de *2,4GHz*, conectada a estação (*ap1*). No momento em que foram parados, os carros estavam trafegando no SUMO com uma velocidade pré estabelecida de 20 *Km/h*.

Para comparar o comportamento da rede utilizando única e múltipla tabela, são feitos experimentos semelhantes em ambas as estruturas. O aplicativo *iperf* é utilizado para realizar as conexões entre o *ap1* e as estações móveis (carros). Com a estrutura em funcionamento, foi atribuído ao *switch* o controlador equipado com a tabela padrão *OpenFlow* (única tabela), e também com o controlador equipado com múltiplas tabelas de fluxo. Previamente estruturado, este controlador está equipado e, portanto passa a inserir as funções *Firewall*, Encaminhamento, *DHCP* e *QoS*. Após a execução do experimento, os resultados foram coletados e passam a ser exibidos e discutidos a seguir.

5.3.2 Resultados - *jitter*

Assim como ocorreu no cenário vCPE, para a primeira rodada de testes a métrica avaliada foi o *jitter*. Conforme já discutido na Seção 5.2.2, o *jitter* pode ser definido como a medida de variação do atraso entre os pacotes sucessivos de dados trafegando na rede. Quando o *jitter* é muito grande a rede perde desempenho e ocorre o atraso na entrega do pacote [1]. As Tabelas 5.20 e 5.21 apresentam os valores de *jitter* com a Variação do percentual por *hosts*, registrados após a execução dos experimentos nos cenários Sd-VANETs equipados com UTF e MTF.

Tabela 5.20: Jitter com 01 Fluxo

	UTF	MTF
Valor médio	0,35	0,30
Variação Percentual	17,56%	

Tabela 5.21: Jitter com 02 Fluxos

	UTF	MTF
Valor médio	0,93	0,88
Variação Percentual	5,98%	

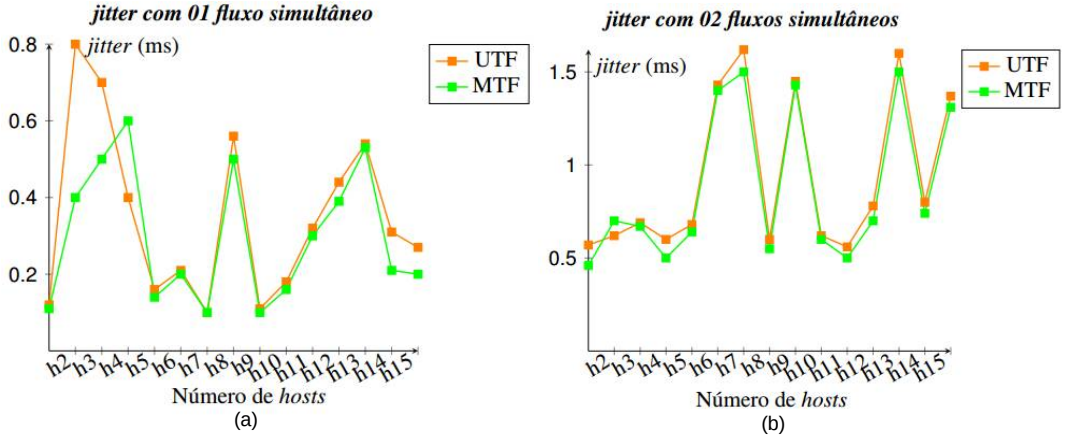

Figura 5.15: *Variação do atraso (jitter) com 01 e 02 Fluxos simultâneos*

Conforme apresentado na Figura 5.15, disferindo apenas 01 fluxo simultaneamente por *host*, os pacotes sofrem uma variação de atraso muito pequena no primeiro nó (h2). Chega a subir sete posições no h3 (0,8 *ms*), e começa a oscilar a partir do nó h4, ficando entre 0,1 *ms* e 0,6 *ms*.

Para o *jitter* com 02 fluxos simultâneos, os primeiros nós ficam estáveis, sofrendo um atraso na entrega de pouco mais de 0,5 ms. A partir do host h6 esse atraso passa a ocorrer entre 0,6 *ms* e 1,6 *ms*. Comparado com o experimento enviando 01 fluxo simultâneo, a rede com 02 fluxos sofre um aumento natural do tempo de atraso.

Tabela 5.22: Jitter com 10 Fluxos

	UTF	MTF
Valor médio	0,35	0,30
Variação Percentual	17,56%	

Tabela 5.23: Jitter com 20 Fluxos

	UTF	MTF
Valor médio	0,93	0,88
Variação Percentual	5,98%	

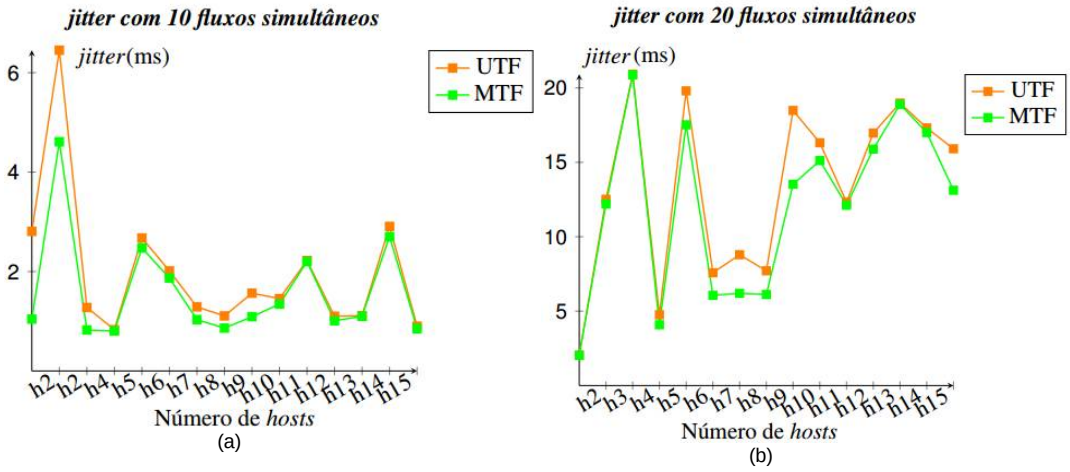

Figura 5.16: *Variação do atraso (jitter) com 10 e 20 Fluxos simultâneos*

As Tabelas 5.22 e 5.23 apresentam os resultados do *jitter* com os valores médios de 1865 *ms* para UTF e 1365 *ms* para MTF, ambos aferidos com 10 fluxos. E com 20 fluxos as médias foram 13361 *ms* na estrutura UTF e 12051 *ms* na estrutura MTF. Com essas médias, conseguiu-se atingir uma Variação do Percentual de 36,63% com 10 fluxos e 10,86% para 20 fluxos simultâneos. A Figura 5.16 (a) apresenta o gráfico para o *jiiter* com 10 fluxos simultâneos obtidos com as estruturas UTF e MTF. A variação do atraso na entrega dos pacotes começa em torno de 2 a 3 *ms* para o *h2*. No *h3* a estrutura equipada com UTF chega a atingir 6 *ms*. Sofre uma queda na sequencia e fica em torno de 2 ms para todos os *hosts*.

Com o envio de 20 fluxos simultâneos a estrutura começa baixa, mas logo assume um *jitter* em torno de 20 *ms* a partir do *host* h4. Na sequencia oscila entre 10 e 16 *ms*. Esse comportamento também é esperado, tendo em vista que com um número maior de fluxos simultâneos disferidos em um único nó, a rede tende a aumentar o tempo da variação do atraso na entrega dos pacotes.

5.3.3 Resultados - *Throughput*

Conforme já especificado na Seção 5.2.3, a vazão (*throughput*), é a quantidade de dados isentos de erros transferidos com sucesso entre dois nós por unidade de tempo. As Tabelas 5.24 e 5.25 apresentam os resultados do *throughput* com 1 e 2 fluxos, com 1642,6 *Kbps* para UTF e 1673,6 *Kbps* para MTF. De acordo com a tabela, a variação percentual com 01 fluxo ficou em 1,88%, e para 02 apresentou uma redução, ficando em 7,67%.

A Figura 5.17 apresenta o gráfico para 1 e 2 fluxos. Da mesma maneira que foi especificado para o cenário vCPE, para este experimento também foi definido que a

rede poderá atingir até 2 megabits por segundo (*Mbits/s*). A aferição do *throughput* foi inicialmente realizada utilizando 01 fluxo. Na sequencia, foram realizados testes com 2, 10 e 20 fluxos.

Tabela 5.24: *Throughput com 01 Fluxo*

	UTF	MTF
Valor médio	1642,6	1673,6
Variação Percentual	1,88%	

Tabela 5.25: *Throughput com 02 Fluxos*

	UTF	MTF
Valor médio	1542,6	1660,9
Variação Percentual	7,67%	

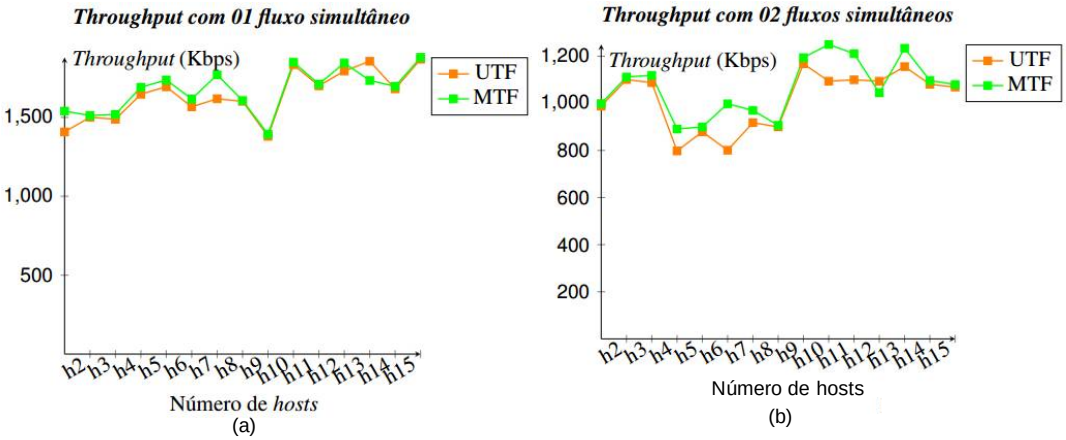

Figura 5.17: *Vazão com 01 e 02 Fluxos simultâneos*

Os resultados são apresentados na Figura 5.17 (a) para um *throughput* utilizando 01 fluxo simultâneo, disparado por todos as 15 estações (carros) da rede. A taxa começa em 1.500 *Kbps* e mantém uma oscilação natural até o último fluxo, chegando a atingir uma taxa de 2 *Mbps*. Com 02 fluxos simultâneos a taxa passa de 1.500 *Kbps* e se mantém bem próximo aos 2 *Mbps* durante todo o restante da conexão. As taxas com 01 e com 02 fluxos são bem parecidas para as duas estruturas propostas, equipadas com UTF e MTF.

Tabela 5.26: *Throughput com 10 Fluxos*

	UTF	MTF
Valor médio	1012,2	1043,5
Variação Percentual	3,09%	

Tabela 5.27: *Throughput com 20 Fluxos*

	UTF	MTF
Valor médio	610,1	750,1
Variação Percentual	22,94%	

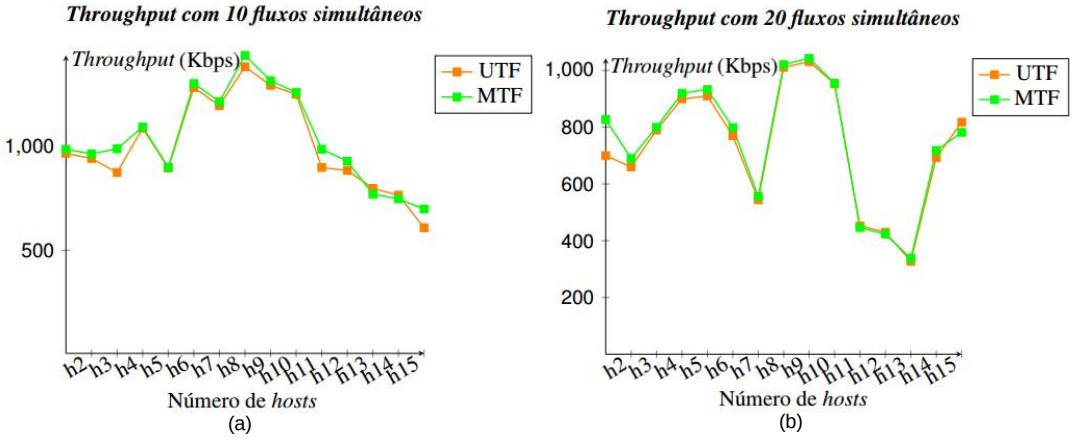

Figura 5.18: *Vazão com 10 e 20 Fluxos simultâneos*

A Tabela 5.26 apresenta os resultados do *throughput* para 10 fluxos. Na estrutura UTF a taxa ficou em 1012,2 *Kbps* e 1043,5 *Kbps* na estrutura MTF. Para essas médias, a variação percentual foi de 3,09%. De acordo com a Figura 5.18 (a), com 10 fluxos simultâneos, a taxa começa com valores próximos a 1 *Mbps*, sobe um pouco, e a partir da estação de número 10 cai, chegando a encostar no valor de 500 *Kbps*.

Com 20 fluxos, a Tabela 5.27 apresenta 610,1 *Kbps* para a estrutura UTF e 750,1 *Kbps* para a estrutura MTF. O Gráfico da Figura 5.18 (b) apresenta os valores com uma baixa vazão de dados (*throughput*). Utilizando 20 fluxos simultâneos, os valores chegam a ficar bem próximos de 200 *Kbps*. O que é considerado ruim em uma rede que pode atingir até 2 *Mbps*, mas é uma situação esperada, tendo em vista que um grande número de conexões simultâneas tende a apresentar uma baixa vazão.

5.3.4 Resultados - Perdas de pacotes

Na Seção 5.2.4 discutimos que a perda de pacotes pode afetar qualquer aplicativo, sendo mais provável a interrupção daqueles que dependem de transferências de dados em tempo real, como programas que transmitem áudio ou vídeo, juntamente com outras várias plataformas de sistemas digitais [2]. Uma Sd-VANET deve também oferecer suporte a esse tipo de serviço. A partir da Tabela 5.28 e da Figura 5.19, é possível analisar os pacotes de dados perdidos, durante os experimentos utilizando as estruturas UTF e MTF.

Tabela 5.28: *Perdas de pacotes, 01 Fluxo*

	UTF	MTF
Valor médio	0,22	0,10
Variação Percentual	114,71%	

Tabela 5.29: *Perda de pacotes, 02 Fluxos*

	UTF	MTF
Valor médio	0,36	0,32
Variação Percentual	16,84%	

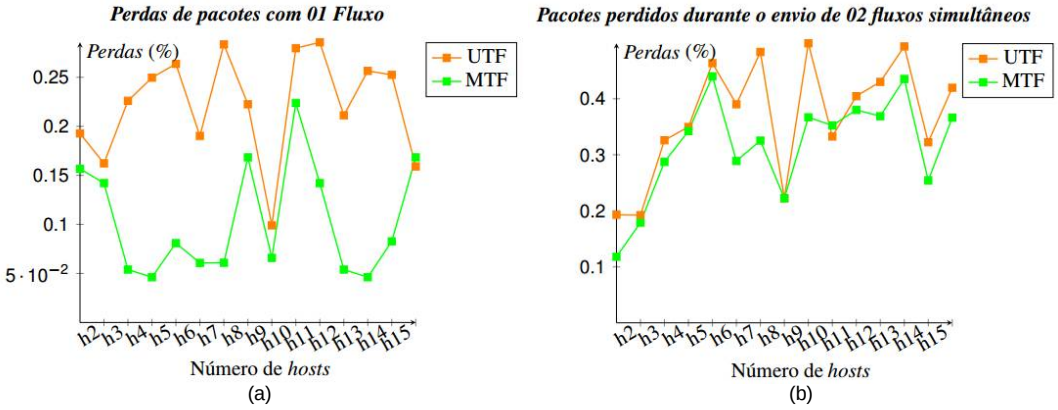

Figura 5.19: *Pacotes perdidos com 01 e 02 Fluxos simultâneos*

A Tabela 5.28 apresenta os resultados para as perdas de pacotes com 01 fluxo. A Variação Percentual ficou em 114%. O Gráfico na Figura 5.19 apresenta as perdas de pacotes para as estruturas UTF e MTF. De forma clara é possível perceber que a estrutura representada pela linha de cor laranja (UTF) se mantém maior em relação a estrutura representada pela linha de cor verde (MTF). Ou seja, as perdas de pacotes, que em UTF oscilam entre *0,2% a 0,3%*, em MTF se observa uma oscilação menor *0,15% a 0,1%*, que além de ser quase insignificante para a rede, revela que a estrutura MTF com 01 fluxo simultâneo se mantém melhor que a que utiliza UTF.

A Tabela 5.29 apresenta o resultados para UTF e MTF com 02 fluxos. A Variação do Percentual é de 16,84%. Com 02 fluxos simultâneos, as duas estruturas se tornam bem semelhantes entre si. As perdas estão aumentando de forma gradual, começando em torno de *0,2%* e atingindo picos de *0,5%* de perdas totais de pacotes.

Tabela 5.30: *Perdas de pacotes, 10 Fluxos*

	UTF	MTF
Valor médio	1802	1817
Variação Percentual	0,86%	

Tabela 5.31: *Perda de pacotes, 20 Fluxos*

	UTF	MTF
Valor médio	13,80	13,30
Variação Percentual	3,66%	

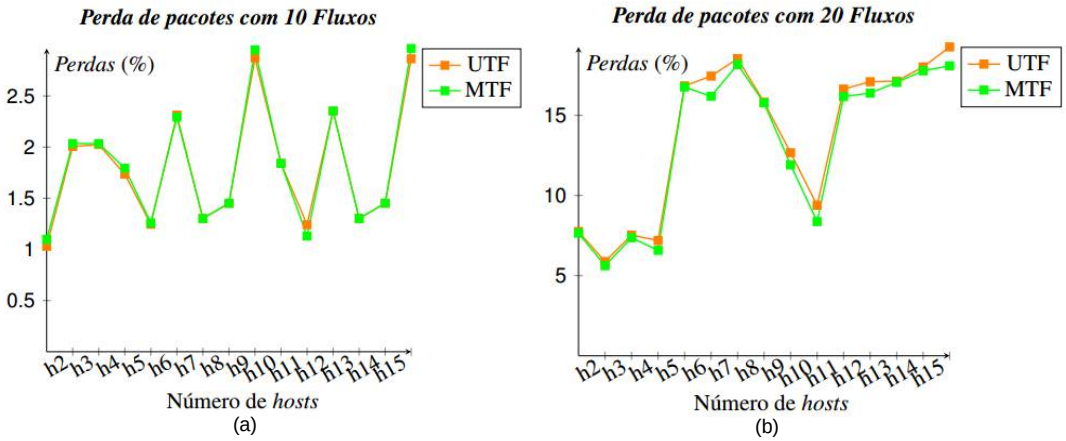

Figura 5.20: *Pacotes perdidos com 10 e 20 Fluxos simultâneos*

As Tabelas 5.30 e 5.31 apresentam os resultados para 10 e 20 fluxos simultâneos. Tem uma Variação Percentual de 0,86% com 10 fluxos e 3,66% para 20 fluxos. De acordo com o gráfico da Figura 5.20, os fluxos dos nós 2 ao 15 das duas estruturas, sofrem importantes perdas de pacotes, de *0,2% a quase 3%* do total de seus pacotes, utilizando 10 fluxos simultâneos. Com 20 fluxos simultâneos, a rede chega a perder até *17%* do total de seus pacotes.

5.3.5 Discussão dos Resultados - Sd-VANET

As medidas de *jitter* apresentados para as redes UTF e MTF utilizando 01 e 02 fluxos revelaram valores aceitáveis para uma rede sem fio com as características de uma Sd-VANET. O mesmo ocorreu para 10 e 20 fluxos simultâneos, mesmo que os valores tenham subido bastante, especialmente quando se utilizou 20 fluxos simultâneos. Entretanto, em todas os testes a estrutura MTF se mostrou com valores menores de *jitter*. O que confirma que quando a estrutura utiliza o esquema de múltiplas tabelas de fluxo, a flexibilidade da rede proporciona um melhor desempenho geral em comparação com a estrutura tradicional, equipada com única tabela de fluxo. A Variação Percentual Média

do jitter foi de 17,77% entre as duas estruturas. Isso significa que a estrutura MTF obteve quase 18% a menos de variação de atraso na entrega de pacotes. A fórmula utilizada para chegar a esse resultado é a mesma utilizada no cenário vCPE, apresentada na seção 5.2.5.

O *Throughput* das redes UTF e MTF utilizando 01 e 02 fluxos simultâneos apresentaram valores já esperados para uma rede sem fio. Foi definido que a quantidade máxima de dados isentos de erros transferidos com sucesso entre as estações, não poderia ultrapassar os 2 *Mbps*. E de acordo com os Gráficos presentes na Figura 5.17 é exatamente isso o que acontece. Utilizando 10 e 20 fluxos simultâneos a rede tem essa taxa reduzida pra quase 200 *Kbps*. O que também já é esperado para uma rede sem fio nas condições apresentadas. O ponto principal do experimento é que em todos os casos, a estrutura equipada com MTF se mantém com as taxas superiores as da estrutura UTF. A Variação Percentual Média da MTF foi de 8,9% em relação a estrutura UTF. O que prova que a rede estruturada com MTF apresenta um desempenho superior em relação à UTF.

Por último, as perdas de pacotes foram avaliadas com 01 e 02 fluxos simultâneos para as estruturas UTF e MTF. De acordo com o que se apresentou no gráfico da Figura 5.19, com 01 fluxo as perdas da estrutura UTF foi bem diferente da estrutura MTF. Com 10 e 20 fluxos simultâneos, mais uma vez a estrutura MTF se mostrou superior. O que também pode ser comprovado com o resultado da Variação Percentual Média, que foi de 33,59%.

CAPÍTULO 6

Conclusões e Trabalhos Futuros

Este trabalho apresentou a implementação de múltiplas tabelas de fluxo em *switches OpenFlow*, e utilizou os cenários vCPE e Sd-VANET para avaliar essa implementação, comparando com os mesmos cenários dispostos em *switches* equipados com única tabela de fluxo. Para atingir esse objetivo, foi necessário compreender os conceitos envolvidos na construção de uma SDN. E as Redes Definidas por *Software* são um novo paradigma que tornam possível o desenvolvimento de diversas aplicações, inclusive nas áreas de vCPE e Sd-VANETs. Apresentamos os conceitos desses novos paradigmas e discutimos que a virtualização das estruturas causam um importante impacto monetário, especialmente na redução dos custos operacionais e de aquisição de material (*OPEX e CAPEX*), e as empresas de telecomunicações tem tirado grande proveito dessas vantagens.

A medida em que as pesquisas avançam, o protocolo *OpenFlow* está em constante desenvolvimento, e o correto entendimento dos planos de controle e de dados que esboçam as estruturas vCPE e SD-VANETs, é de fundamental importância para o avanço das pesquisas e o acompanhamento simétrico da crescente demanda tecnológica. Além do mais, a discussão em torno da correta instanciação do *switch OpenFlow*, revelou que o estado da arte já conta com bastantes contribuições em torno de aplicações do tipo vCPE, em especial, por estarem se beneficiando das vantagens oferecidas pelo o esquema de múltiplas tabelas de fluxo. As contribuições que existem para as Sd-VANETS também estão em alta. Muitos trabalhos abordam o tema e apresentam diversos tipos de soluções, rompendo assim vários desafios impostos por esse novo paradigma. Entretanto, o estado da arte não aborda a implementação de múltiplas tabelas nos vCPEs utilizando nós *wireless*. Muito menos apresentam a implementação das Sd-VANETs com suas funções distribuídas no esquema de múltiplas tabelas de fluxo.

A principal contribuição deste trabalho é a de propor a implementação e avaliação de dois cenários de rede, que buscam se aproveitar do esquema de múltiplas tabelas de fluxo, deixando as estruturas bem mais flexíveis e interoperáveis. O primeiro cenário reproduz e virtualiza CPE com múltiplas tabelas. O segundo implementa uma Sd-VANET também beneficiada com o uso das múltiplas tabelas de fluxo. Inicialmente criamos os ce-

nários vCPE e Sd-VANET equipados com apenas 01 (um) *switch OpenFlow*, utilizando o controlador RYU como sistema operacional da rede. Vale lembrar que a utilização do RYU nos permitiu tirar proveito de sua capacidade de integração com o protocolo *OpenFlow*, tendo como consequência a possibilidade de se criar múltiplas funções de rede virtualizadas no *switch OpenFlow*. Uma vez definidos os cenários e seus parâmetros, os resultados puderam ser corretamente avaliados através da comparação dos mesmos cenários (vCPE e Sd-VANET) implementados com *switches* equipados com múltiplas tabelas de fluxo (MTF).

Realizar a manipulação do *switch OpenFlow* com o esquema de múltiplas tabelas de fluxo, faz com que as estruturas vCPE e SD-VANETs possam atender aos atuais e crescentes requisitos de interoperabilidade exigidos pelo mercado. Esses cenários se beneficiam do modelo de múltiplas tabelas de fluxo porque a inserção de múltiplas funções, como é característico das duas estruturas, fica melhor distribuída e torna o ambiente mais flexível, deixando a resolução de problemas mais rápido.

Os experimentos utilizando MTF, comparadas com as estruturas equipadas com UTF apresentaram resultados satisfatórios. No cenário vCPE obtivemos um ganho de 14,88% no desempenho geral da estrutura com MTF. No cenário Sd-VANET esse ganho foi de 20,09% em relação ao mesmo cenário equipado UTF. De maneira geral, podemos dizer que os dois cenários juntos tiveram um desempenho 17,48% maior em comparação com os mesmos cenário equipados com única tabela. Constatamos com isso que o uso de múltiplas tabelas cria possibilidades de crescimento das estruturas sem a preocupação com o estouro da memória decorrente do numero de entradas na tabela de fluxo. Com isso, os vCPEs foram implementados com as funções *Firewall*, encaminhamento, *DHCP* e *QoS*, com seus respectivos nós sem fio, e por meio do emulador de redes Mininet-WiFi foi possível realizar a comunicação satisfatória com o plano de controle da rede.

Utilizando as múltiplas tabelas nas SD-VANETs, foi possível ativar uma função de segurança que conseguiu interromper um fluxo originado de um nó específico, percebemos com isso que é possível obter um ganho importante na criação de funções de segurança nas Sd-VANETs, por exemplo, em situações onde veículos de apoio precisam abrir caminho para atender uma chamada de emergência, vindas de centrais de comunicação conectadas as RSUs, distribuídas no decorrer da estrada. Com as múltiplas tabelas é possível o controlador *OpenFlow* inserir funções de encaminhamento de pacotes, outras regras de *Firewall*, mensagens de alerta específicos, etc. Os projetos de construção de infraestruturas baseadas em SD-VANETs podem aumentar de forma significativa suas possibilidades ao utilizarem o modelo de *switches* equipados com múltiplas tabelas de fluxo.

6.1 Trabalhos Futuros

Como trabalhos futuros, pretendemos estender a funcionalidade dos *switches* equipados com múltiplas tabelas de fluxo para agregar dispositivos de *hardware* específicos, tais como câmeras, sensores de temperatura capazes de fornecer um relatório de saúde, onde se consiga até mesmo diagnosticar um problema baseado nas informações disponíveis em um banco de dados armazenado em um local também manipulado pelo controlador RYU. Além disso, pretendemos melhorar a capacidade de programação do *switch* com outros sistemas. E a linguagem de programação P4 é um forte candidato para isso. Também, com o objetivo de agregar e aumentar a largura de banda, pode-se criar redes sem fio com seus nós equipados com múltiplas interfaces capazes de fornecer backup e outras funcionalidades a rede. E as interfaces do tipo *binding* e *bonding* também são fortes candidatas para esse tipo de funcionalidade.

Um outro trabalho que pode ser desenvolvido com a utilização do esquema de múltiplas tabelas de fluxo, é a implementação de *gateways* com o objetivo de realizar a interação entre o ambiente virtual e o real. Com isso, o uso das SDNs poderão ser aplicadas de maneiras ainda mais práticas e objetivas. Na mesma linha de pensamento, a implementação de *handover* para realizar a transição de estações entre as células, sejam elas WiFi, LTE ou qualquer outra tecnologia sem fio, também poderão se beneficiar do esquema de funções distribuídas em múltiplas tabelas de fluxo. Enfim, existem inúmeras possibilidades de se tirar proveito da flexibilidade oferecida, não somente pelos paradigmas SDN e NFV, mas especialmente pelo uso adequado do protocolo OpenFlow, juntamente com sua atual especificação implementando o uso de múltiplas tabelas, na construção dos ativos centralizadores de rede baseados no seu protocolo.

Pode-se ainda realizar trabalhos baseados em experimentos utilizando *switches OpenFlow* equipados com múltiplas tabelas de fluxo, visando tornar as estruturas SDN cada vez mais flexíveis e interoperáveis. Tais trabalhos podem gerar soluções importantes capazes de produzir economias para os governos por, no caso das Sd-VANETs, gerar uma melhor gestão do trânsito nas cidades e diminuir os casos de acidentes e consequentemente diminuírem as despesas com hospitais das possíveis vítimas. E no caso das vCPEs, trazer mais comodidade e segurança para o crescente número de usuários que utilizam os mais variados tipos de serviços de compartilhamento de dados.

Referências Bibliográficas

[1] **Metric ieee standard for measurement of video jitter and wander.** *Measurement Metric IEEE Std 1521-2003*, p. c1–14, 2009.

[2] AFRIZAL, G.; HENDRAWAN. **Impact of random and burst packet loss on voice codec g.711, g.722, g.729, amr-nb, amr-wb.** In: *2018 4th International Conference on Wireless and Telematics (ICWT)*, p. 1–4, 2018.

[3] AMARO GALHARDO, E.; OLIVEIRA, A. C. **iclassmft: Proposed multiple flow tables classes to integrate security and flexibility into sdn switches.** In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, p. 1422–1427, 2019.

[4] ARIF, M.; WANG, G.; GEMAN, O.; BALAS, V.; TAO, P.; BREZULIANU, A.; CHEN, J. **Sdn-based vanets, security attacks, applications, and challenges.** *Applied Sciences*, 10:3217, 05 2020.

[5] BAZZAN, A. L. C.; KLÜGL, F. **Introduction to Intelligent Systems in Traffic and Transportation.** 2014.

[6] BLENK, A.; BASTA, A.; REISSLEIN, M.; KELLERER, W. **Survey on network virtualization hypervisors for software defined networking.** *IEEE Communications Surveys Tutorials*, 18(1):655–685, 2016.

[7] BREIKI, M. S. A.; ZHOU, S.; LUO, Y. R. **Development of openflow native capabilities to optimize qos.** In: *2020 Seventh International Conference on Software Defined Systems (SDS)*, p. 67–74, 2020.

[8] BRUZGIENE, R.; NARBUTAITE, L.; ADOMKUS, T. **Manet network in internet of things system.** In: Ortiz, J. H.; de la Cruz, A. P., editors, *Ad Hoc Networks*, chapter 5. IntechOpen, Rijeka, 2017.

[9] CANO, J.; CANO, J.; TOH, C. K.; CALAFATE, C. T.; MANZONI, P. **Easymanet: an extensible and configurable platform for service provisioning in manet environments.** *IEEE Communications Magazine*, 48(12):159–167, 2010.

[10] CHAHAL, M.; HARIT, S. **Towards software-defined vehicular communication: Architecture and use cases.** In: *2017 International Conference on Computing, Communication and Automation (ICCCA)*, p. 534–538, 2017.

[11] CHEN, Z.; WU, Y.; GE, J.; YUEPENG, E. **A new lookup model for multiple flow tables of open flow with implementation and optimization considerations.** In: *2014 IEEE International Conference on Computer and Information Technology*, p. 528–532, 2014.

[12] CHI, P.; WANG, M.; GUO, J.; LEI, C. **Sdn migration: An efficient approach to integrate openflow networks with stp-enabled networks.** In: *2016 International Computer Symposium (ICS)*, p. 148–153, 2016.

[13] DOS REIS FONTES, R.; CAMPOLO, C.; ESTEVE ROTHENBERG, C.; MOLINARO, A. **From theory to experimental evaluation: Resource management in software-defined vehicular networks.** *IEEE Access*, 5:3069–3076, 2017.

[14] ELNASHAR, A.; EL-SAIDNY, M. A.; SHERIF, M. **Deployment Strategy of LTE Network**, p. 273–348. 2014.

[15] FONTES, R. R.; AFZAL, S.; BRITO, S. H. B.; SANTOS, M. A. S.; ROTHENBERG, C. E. **Mininet-wifi: Emulating software-defined wireless networks.** In: *Network and Service Management (CNSM), 2015 11th International Conference on*, p. 384–389, Nov 2015.

[16] GARAI, M.; REKHIS, S.; BOUDRIGA, N. **Communication as a service for cloud vanets.** In: *2015 IEEE Symposium on Computers and Communication (ISCC)*, p. 371–377, 2015.

[17] GE, J.; CHEN, Z.; WU, Y.; E, Y. **H-soft: a heuristic storage space optimisation algorithm for flow table of openflow.** *Concurrency and Computation: Practice and Experience*, 27(13):3497–3509, 2015.

[18] GHORI, M. R.; ZAMLI, K. Z.; QUOSTHONI, N.; HISYAM, M.; MONTASER, M. **Vehicular ad-hoc network (vanet): Review.** In: *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, p. 1–6, 2018.

[19] GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M.; MCKEOWN, N.; SHENKER, S. **Nox: Towards an operating system for networks.** *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[20] GURTOV, A. **Middlebox Traversal**, p. 161–180. 2008.

[21] HARTENSTEIN, H.; LABERTEAUX, K. **Introduction**, p. 1–19. 2010.

[22] HE, Z.; CAO, J.; LIU, X. **Sdvn: enabling rapid network innovation for heterogeneous vehicular communication.** *IEEE Network*, 30(4):10–15, 2016.

[23] FLOODLIGHT, Q. W. **Floodlight Homepage - http://floodlight.OpenFlowhub.org/.** Technical report, FloodLight Repository, Acessado em Março de 2019 - (On Line), 2019.

[24] NOX, N. N. . **NOX Homepage - https://noxrepo.github.io/pox-doc/html - (Nicira Networks; Stanford University and International Computer Science Institute, UC Berkeley).** Acessado em Março de 2019, (On Line), 2019.

[25] CISCO, C. N. **Online Post - Acessado em janeiro de 2020.** Technical report, Cisco Networking, On Line, 2020.

[26] MULTIPLEFLOWTABLE, O. F. **The Benefits of Multiple Flow Tables and TTPs - Acessado em junho de 2019.** Technical report, The Open Networking Foundation, On Line, 2020.

[27] ONF, A. N. **Online User's Guide - Acessado em março de 2020.** Technical report, The Open Network Foundation, On Line, 2020.

[28] ONOS, T. V. **ONOS Homepage - https://www.opennetworking.org/onos/.** Acessado em Março de 2019 - (On Line), 2019.

[29] HUANG, N.; LI, C.; CHEN, C.; HSU, I.; LI, C.; CHEN, C. **A novel vcpe framework for enabling virtual network functions with multiple flow tables architecture in sdn switches.** In: *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, p. 64–69, 2017.

[30] HUSSEIN, A.; ELHAJJ, I. H.; CHEHAB, A.; KAYSSI, A. **Sdn vanets in 5g: An architecture for resilient security services.** In: *2017 Fourth International Conference on Software Defined Systems (SDS)*, p. 67–74, 2017.

[31] IPERF. **iperf - the ultimate speed test tool for tcp, udp and sctp.** www.iperf.fr/~iperf, 2020.

[32] KAUR, R.; SINGH, T. P.; KHAJURIA, V. **Security issues in vehicular ad-hoc network(vanet).** In: *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, p. 884–889, 2018.

[33] KAUR, R.; SINGH, T. P.; KHAJURIA, V. **Security issues in vehicular ad-hoc network(vanet).** In: *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, p. 884–889, 2018.

[34] KAUR, S.; KAUR, K.; GUPTA, V. **Implementing openflow based firewall.** In: *3rd International Conference on Electrical, Electronics, Engineering Trends, Communication, Optimization and Sciences (EEECOS 2016)*, p. 1–3, 2016.

[35] KIM, E.; LEE, S.; CHOI, Y.; SHIN, M.; KIM, H. **A multiple flow tables construction scheme for service function chaining in nfv.** In: *2015 International Conference on Information and Communication Technology Convergence (ICTC)*, p. 113–115, 2015.

[36] KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. **Software-defined networking: A comprehensive survey.** *Proceedings of the IEEE*, 103(1):14–76, 2015.

[37] KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. **Software-defined networking: A comprehensive survey.** *Proceedings of the IEEE*, 103(1):14–76, 2015.

[38] KU, I.; LU, Y.; GERLA, M.; GOMES, R. L.; ONGARO, F.; CERQUEIRA, E. **Towards software-defined vanet: Architecture and services.** In: *2014 13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)*, p. 103–110, 2014.

[39] LANTZ, B.; HELLER, B.; MCKEOWN, N. **A network in a laptop: Rapid prototyping for software-defined networks.** In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, New York, NY, USA, 2010. Association for Computing Machinery.

[40] LIU, Y.; CHEN, C.; CHAKRABORTY, S. **A software defined network architecture for geobroadcast in vanets.** In: *2015 IEEE International Conference on Communications (ICC)*, p. 6559–6564, 2015.

[41] MARTINEZ, F. J.; TOH, C.; CANO, J.; CALAFATE, C. T.; MANZONI, P. **Emergency services in future intelligent transportation systems based on vehicular communication networks.** *IEEE Intelligent Transportation Systems Magazine*, 2(2):6–20, 2010.

[42] MATIAS, J.; GARAY, J.; TOLEDO, N.; UNZILLA, J.; JACOB, E. **Toward an sdn-enabled nfv architecture.** *IEEE Communications Magazine*, 53(4):187–193, 2015.

[43] MATSUMOTO, N.; HAYASHI, M. **Performance improvement of flow switching with automatic maintenance of hash table assisted by wildcard flow entries.** In: *The 10th International Conference on Optical Internet (COIN2012)*, p. 12–13, 2012.

[44] MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. **Openflow: Enabling innovation in campus networks.** *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[45] MINOLI, D. **Internet of Things Definitions and Frameworks**, p. 28–47. 2013.

[46] NGUYEN, T.; NGUYEN, T.; YOO, M. **Analysis of deployment approaches for virtual customer premises equipment.** In: *2018 International Conference on Information Networking (ICOIN)*, p. 289–291, 2018.

[47] PABLO, M. **Conectando veiculo para tudo. https://www.embarcados.com.br.** *Embaracados*, p. OnLine – Acessado em Setembro, 2020.

[48] PENTTINEN, J. T. J. **Protocols**, p. 73–99. 2013.

[49] PEREZ, K. G.; SCOTT-HAYWARD, S.; YANG, X.; SEZER, S. **Memory cost analysis for openflow multiple table lookup.** In: *2015 28th IEEE International System-on-Chip Conference (SOCC)*, p. 322–327, 2015.

[50] PRESUHN, R. **Version 2 of the protocoloperations for the simple networkmanagement protocol (snmp).** Internal report, Internet Engineering Task Force, Jun 2002.

[51] PUJOLLE, G. **Fabric, SD-WAN, vCPE, vRAN, vEPC**, p. 33–50. 2020.

[52] PUJOLLE, G. **New-generation Protocols**, p. 115–132. 2020.

[53] RODRIGUEZ, J. **Cooperation for Next Generation Wireless Networks**, p. 105–124. 2014.

[54] RYUPROJECTTEAM, R. **RYU SDN Framework - English Edition.** Release 1.0. RYU project team, 2014.

[55] SALAHUDDIN, M. A.; AL-FUQAHA, A.; GUIZANI, M. **Software-defined networking for rsu clouds in support of the internet of vehicles.** *IEEE Internet of Things Journal*, 2(2):133–144, 2015.

[56] SCHENKER, S. **The future of networking, the past of protocols.** Acessado em fev. de 2020, Disponibilizado no youtube em: www.youtube.com/watch?v=YHeyuD89n1Y, Oct 2011.

[57] SHAH, S. A.; FAIZ, J.; FAROOQ, M.; SHAFI, A.; MEHDI, S. A. **An architectural evaluation of sdn controllers.** In: *2013 IEEE International Conference on Communications (ICC)*, p. 3504–3508, 2013.

[58] SOUA, R.; KALOGEITON, E.; MANZO, G.; DUARTE, J.; PALATTELLA, M.; DI MAIO, A.; BRAUN, T.; ENGEL, T.; VILLAS, L.; RIZZO, G. **SDN Coordination for CCN and FC Content Dissemination in VANETs**, p. 221–233. 12 2017.

[59] SUZUKI, H.; AGAWA, Y.; KUBO, R. **Ryu sdn framework: Open-source sdn platform software.** In: *2014 Ryu sdn framework: Open-source sdn platform software (Technical Review)*, volume 12, p. 1–5, 2014.

[60] TAYYAB, M.; GELABERT, X.; JÄNTTI, R. **A simulation study on handover in lte ultra-small cell deployment: A 5g challenge.** In: *2019 IEEE 2nd 5G World Forum (5GWF)*, p. 388–392, 2019.

[61] TECHNOLOGIES QUALCOMM, QUALCOMM. **Webinar presentation, qualcomm-tech.** In: SP Patil, S. P., editor, *How NR based sidelinkexpands 5G C-V2X to support new advanced use cases*, p. 1–27. Qualcomm Technologies, Inc, 2020.

[62] TRUONG, N. B.; LEE, G. M.; GHAMRI-DOUDANE, Y. **Software defined networking-based vehicular adhoc network with fog computing.** In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, p. 1202–1207, 2015.

[63] VIRIYASITAVAT, W.; BOBAN, M.; TSAI, H.; VASILAKOS, A. **Vehicular communications: Survey and challenges of channel and propagation models.** *IEEE Vehicular Technology Magazine*, 10(2):55–66, 2015.

[64] WANG, H.; CHEN, C.; LU, S. **An sdn-based nat traversal mechanism for end-to-end iot networking.** In: *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, p. 1–4, 2019.

[65] WANG, J.; CHEN, Y. **An sdn-based defensive solution against dhcp attacks in the virtualization environment.** In: *2017 IEEE Conference on Dependable and Secure Computing*, p. 529–530, 2017.

[66] YADAV, N.; CHUG, U. **Secure routing in manet:a review.** In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, p. 375–379, 2019.

[67] ZHANG, Y. **Network Function Virtualization**, p. 37–65. 2018.

[68] ZHENG, K.; HOU, L.; MENG, H.; ZHENG, Q.; LU, N.; LEI, L. **Soft-defined heterogeneous vehicular network: architecture and challenges.** *IEEE Network*, 30(4):72–80, 2016.