REDES DEFINIDAS POR SOFTWARE COM SERVIÇOS ORIENTADOS A NOMES

LUCAS AUGUSTO MAIA DA SILVA

REDES DEFINIDAS POR SOFTWARE COM SERVIÇOS ORIENTADOS A NOMES

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Marcos Augusto Menezes Vieira
Coorientador: Dorgival Guedes

Belo Horizonte
Outubro de 2016

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Silva, Lucas Augusto Maia da.

S586r Redes definidas por software com serviços orientados a nomes. / Lucas Augusto Maia da Silva. – Belo Horizonte, 2016.
xxiii, 98 f.: il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Marcos Augusto Menezes Vieira.
Coorientador: Dorgival Olavo Guedes Neto.

1. Computação - Teses. 2. Redes de computadores. 3. Redes definidas por software. 4. OpenFlow. 5. Nomes de domínio na Internet. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*22(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Redes definidas por software com serviços orientados a nomes

LUCAS AUGUSTO MAIA DA SILVA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. MARCOS AUGUSTO MENEZES VIEIRA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. DORGIVAL OLAVO GUEDES NETO - Coorientador
Departamento de Ciência da Computação - UFMG

PROF. DANIEL FERNANDES MACEDO
Departamento de Ciência da Computação - UFMG

PROF. LUIZ FILIPE MENEZES VIEIRA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 21 de outubro de 2016.

Agradecimentos

Eu gostaria primeiramente de agradecer à minha família que me ajudou muito durante o mestrado. Gostaria de agradecer principalmente ao professor Marcos Augusto por dar todo o suporte durante o mestrado, permitindo a minha participação em vários projetos de temas diversos, sempre me ajudando desde a minha graduação. Professor o meu muito obrigado! Gostaria também de agradecer ao professor Dorgival pelas ideias e ajuda para a nossa pesquisa, elas foram fundamentais para o nosso desenvolvimento.

Gostaria de agradecer ao laboratório Winet, aos professores Daniel Macedo e José Marcos. Além de poder contribuir para o projeto FUTEBOL, foi muito bom ser *sysadmin* do Winet, consegui aprender várias coisas e contribuir para o funcionamento do laboratório. Gostaria também de agradecer ao Lecom, aos colegas de laboratório, aos professores Luiz Filipe, Antônio Otávio e Omar e a chance de participar do projeto Cyber Physical AoP. Foi uma experiência muito boa e com ótimos aprendizados. Acredito que todos os professores envolvidos foram de grande importância neste processo de aprendizado.

A todos os envolvidos, meu muito obrigado!

"the answer my friend,
is blowing in the wind..."
(Bob Dylan)

Resumo

Redes Definidas por Software (SDN) propõe a separação do plano de controle do plano de dados, permitindo um plano de controle programável e logicamente centralizado. SDN trazem inovações e facilitam pesquisas em redes de computadores. Uma das possíveis implementações de uma rede SDN é utilizando o protocolo OpenFlow. Apesar do sucesso, o protocolo OpenFlow possui limitações. O OpenFlow só trabalha com campos de cabeçalhos das camadas L2-L4 para identificação de fluxos. Caso o usuário deseje utilizar dados da camada de aplicação (L7), ele deve buscar outras abordagens, o que na maioria das vezes gera um custo adicional à aplicação (financeiro ou computacional). Este projeto estende o protocolo OpenFlow para atender a camada L7, mais especificamente, lidar com nomes de domínio. O objetivo é aumentar o poder de abstração dos switches OpenFlow. Para isso, estendemos a implementação em software do switch OpenFlow para suportar a criação de regras utilizando nomes de domínio. Nomes de domínio foram escolhidos devido a capacidade de agregar vários endereços IPs em apenas uma regra. Desta forma, nossa solução diminui a quantidade de linhas de código em programas SDN, e a quantidade de regras na tabela de fluxos do switch. Os custos adicionais obtidos foram na vazão, nossa proposta atingiu cerca de 13% a mais que a implementação original, e no consumo total de CPU, nossa solução gastou cerca de 10% a mais que a implementação original. Latência e memória não obtiveram grandes alterações quando comparado com a implementação original. Nosso projeto facilita a criação de aplicações como Firewall, QoS, contadores estatísticos, e outras aplicações utilizando nomes de domínio. Em aplicações que utilizam o controlador para realizar o processamento dos nomes, foi possível diminuir a latência e a quantidade de pacotes enviados ao controlador.

Palavras-chave: Redes de Computadores, Redes Definidas por Software (SDN), OpenFlow, Nomes de Domínio, DNS, Camada de Aplicação.

Abstract

Software Defined Networks (SDN) proposes the separation of the control plane from the data plane, allowing a programmable and logically centralized control plane. SDN brings innovation and ease researches in computer networks. One possible way to implement an SDN architecture is possible with OpenFlow, an open standard to implement SDN systems. Despite its success, OpenFlow has its limitations. OpenFlow only parses fields from layers 2 and 4 (L2-L4), used to identify flows. In cases a user needs to use data from application layer (L7) to identify a flow, it will be necessary to search another approaches. In most cases, these approaches, bring an overhead to the application (financial or computational). This project extends the OpenFlow protocol to work with layer L7, more specifically, to deal with domain names. Our project aims to expand the abstraction power of OpenFlow, providing new matching rules. We extend OpenFlow to support domain names matching rules. We choose domain names because the capability to group many IP addresses in only one domain name. We built a prototype switch and a controller that can handle the domain name extension. Our approach provides a higher abstraction to program the network (reducing the code size), and decreases the amount of rules in the switch flow table. Our solution presents an overhead of 14% on throughput and 10% on CPU in comparison with the original switch implementation. Memory and latency presents similar values in comparison with the original switch implementation. Our system allows to easily implement Firewalls, QoS applications, flow statistic counters, and another applications using domain names. In applications that process domain names in controllers, it was possible to decrease latency and the number of packets sent to controller.

Keywords: Computer Network, Software-Defined Networks (SDN), OpenFlow, Domain Names, DNS, Application Layer.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

1.1 Tema

Redes Definidas por Software (SDN) criaram um novo paradigma para o desenvolvimento em redes de computadores. SDN criam a separação entre os planos de dados e controle criando uma API para comunicação entre eles, isto permitindo a criação de elementos programáveis [ONF, 2014]. Ao realizar esta separação, o plano de dados pode ser desenvolvido focado em desempenho e no encaminhamento eficiente de pacotes.

Uma forma de se implementar uma SDN é através da adoção do protocolo OpenFlow, proposto em 2008 para este objetivo [McKeown et al., 2008]. Um switch OpenFlow opera utilizando tabelas de fluxos. Cada entrada na tabela representa um fluxo que é identificado pelos dados extraídos dos cabeçalhos dos pacotes. Para realizar operações de consulta, inserção, remoção e atualização dos fluxos na tabela, é utilizada a API do protocolo OpenFlow. O controlador é a aplicação responsável pela tomada de decisões sobre os fluxos. Sempre quando novos fluxos chegam ao switch, estes fluxos são encaminhados ao controlador para solicitar qual decisão deve ser tomada. Juniper e HP são exemplos de fabricantes de switches que adotaram o protocolo OpenFlow [Kreutz et al., 2015].

1.2 Problema

Apesar do seu poder e sucesso, o protocolo OpenFlow não está isento de limitações. Em particular, ele apenas opera com cabeçalhos das camadas L2-L4 para a identificação de fluxos [Li et al., 2015]. Portanto, a única forma de extrair dados de cabeçalhos

de pacotes da camada de aplicação (L7), e utilizar esta informação para controlar fluxos é através do uso do controlador SDN para processar os dados desses pacotes. Desta forma, as decisões de encaminhamento são transferidas para o controlador. Esta abordagem leva a perda de desempenho e aumenta a latência da aplicação, devido ao número de mensagens trocadas entre controlador e switch [Mogul et al., 2010], e o tempo de processamento (CPU) gasto pelo controlador. Além disso, ao implementar todas funcionalidades no controlador para rastrear todas as mudanças de estado dos fluxos, pode causar problemas de escalabilidade e desempenho. A complexidade destas implementações é uma das razões para que várias vezes essas funcionalidades são designadas para dispositivos especializados, geralmente identificados como Middleboxes.

Dar mais poder aos switches OpenFlow vêm sendo estudado por grupos filiados à ONF (Open Network Foundation). Novos grupos como o NoviFlow [1] e Freescale [2] propõem extensões ao OpenFlow para trabalhar com a camada de aplicação. Além desses grupos, a Cisco também está investindo em formas de inserir funcionalidades nas camadas L4-L7. Esta abordagem da Cisco é chamada de *service graph* [3] .

1.3 Motivação

A motivação para este trabalho é simplificar as tarefas de rede realizadas pelo controlador de rede estendendo o protocolo OpenFlow. Reduzir o custo e complexidade de programas SDN que utilizam informações da camada de aplicação (nomes de domínio neste caso). Ao fazer com que switches OpenFlow estejam cientes dos nomes de domínio dos fluxos, é possível simplificar programas SDN para que não precisem obter mais essa informação. Esta alteração também reduz a quantidade de mensagens trocadas entre controlador e switches nesses casos, reduzindo a latência do processo como um todo. Ao incluir nomes de domínio à tabela de fluxos é criada a possibilidade de administradores de redes identificarem quais sites são acessados, possibilitando um controle melhor do uso da Internet.

A adição de nomes de domínio à tabela de fluxos de switches OpenFlow trazem os seguintes benefícios para SDN:

Regras de casamento persistentes: regras que utilizam nomes de domínio resistem à mudanças dinâmicas de endereços IPs. Regras tradicionais que utilizam

[1] https://www.opennetworking.org/onf-sdn-solutions-showcase/carrierwan/1645-noviflow

[2] https://www.opennetworking.org/onf-sdn-solutions-showcase/campus/1656-freescale

[3] http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus1000/avs/reference-guide/L4-L7-services/Cisco_AVS_L4-L7_White_Paper.html

[3] http://blogs.cisco.com/datacenter/new-innovations-for-l4-7-network-services-integration-with-ciscos-aci-approach

apenas endereços IP necessitam de atualização a cada mudança de endereço de rede.

Maior grau de abstração: desenvolvedores SDN podem realizar tarefas mais complexas de uma maneira mais simples utilizando uma quantidade menor de código, uma vez que eles podem utilizar diretamente nomes de domínio.

Redução da latência nas aplicações: evitar que o controlador armazene os nomes de domínio melhora o desempenho da aplicação. Além do custo adicional gerado pela troca de mensagens entre controlador e switch, o tempo de processamento gasto pelo controlador pode impactar em aplicações SDN.

Simplificação de serviços prestados por middleboxes: middleboxes são dispositivos de propósito único. Implementá-los em controladores SDN requer que sejam armazenadas e atualizadas as informações sobre os nomes de domínio. Este trabalho pode simplificar o desenvolvimento de um firewall, load balancer, QoS, utilizando nomes de domínio para regras de casamento.

Facilitação do gerenciamento de data centers: em data centers que trabalham com multi inquilinos, administradores de redes podem utilizar esta solução para encaminhar pacotes para os servidores corretos de acordo com os nomes de domínio solicitados. Assim, simplificar o roteamento em redes com vários inquilinos.

Roteamento em redes orientadas a conteúdo: este trabalho pode ser aplicado em CDNs (Content Delivery Networks). O tráfego pode ser encaminhado para os servidores de acordo com o conteúdo requisitado, como imagens, áudio, vídeos, ou streamings ao vivo, caso o nome do servidor refletir o seu tipo de conteúdo. Este roteamento pode ser baseado no artigo "Networking named content" [Jacobson et al., 2009].

Contadores estatísticos para nomes de domínio: as tabelas de fluxo podem ser consultadas de acordo com os nomes de domínio de origem/destino. Assim, é possível recuperar dados estatísticos sobre os domínios acessados.

1.4 Objetivos

Neste trabalho são adicionados dois novos campos na estrutura de casamento do OpenFlow. Os campos adicionados representam os nomes de domínio de origem e destino de cada fluxo. Esta alteração permite a criação de regras de encaminhamento utilizando nomes de domínio, combinada com as ações existentes em switches OpenFlow. Ao fazer isso, controladores SDN não precisariam inspecionar o *payload* de pacotes DNS para obter os nomes de domínio. Assim, é possível eliminar o processamento extra para associar os nomes de domínio com outros campos dos cabeçalhos. Portanto, reduzir o

processamento e latência das aplicações SDN. Com essa funcionalidade é possível em alguns casos eliminar o uso de Middleboxes, especificamente para esse propósito.

O princípio por trás dessa solução é permitir que o controlador crie regras para switches OpenFlow utilizando nomes de domínio de origem/destino associada aos fluxos. Para ser capaz de realizar isso é necessário estender a implementação do protocolo OpenFlow para armazenar os nomes de domínio de origem e destino associados para cada fluxo. O switch OpenFlow deve ser capaz de inspecionar o payload dos pacotes DNS e extrair os nomes de domínio associados. Esta informação é utilizada para construir o mapeamento <IP, Nome de Domínio>. Quando um pacote entra no switch, seu endereço IP é utilizado para acessar este mapeamento. Se um casamento é encontrado, o nome de domínio associado ao pacote é utilizado para realizar o casamento com a tabela do switch estendida. Esses campos podem ser utilizados para criar regras e executar ações.

Nossos objetivos podem ser sintetizados nos seguintes pontos:

Aumentar o poder de abstração dos switches OpenFlow: Os switches OpenFlow utilizam apenas os cabeçalhos dos protocolos das camadas L2-L4. Nosso projeto estende o suporte para o casamento utilizando nomes.

Diminuir a quantidade de regras na tabela do OpenFlow: As regras que utilizam endereços IPs podem ser agregadas em regras utilizando apenas o nome a qual estes IPs pertencem.

Diminuir a latência das aplicações: Ao passar o processamento de pacotes DNS de resposta para o switch a quantidade de pacotes enviados ao controlador diminui, desta forma a latência da aplicação diminui como um todo. Este objetivo é aplicado em casos em que a quantidade de pacotes enviados ao controlador impacte no desempenho das aplicações

1.5 Organização da Dissertação

Esta dissertação está organizada da seguinte forma. O capítulo 2 aborda os conceitos básicos para elaboração deste trabalho. O capítulo 3 apresenta os trabalhos relacionados e realiza o paralelo com a solução proposta. O capítulo 4 apresenta a solução criada, a versão criada do OpenFlow, chamada de DN+OpenFlow. O capítulo 5 discute questões relacionadas à dissertação. O capítulo 6 apresenta a avaliação realizada. Por último, o capítulo 7 encerra este trabalho apresentando as conclusões.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão apresentados os conceitos básicos utilizados para construir esta dissertação. Na primeira seção é apresentado o que é SDN, seus conceitos e sua história. Na segunda seção é apresentado o protocolo OpenFlow, suas características, e versões. Na terceira seção são apresentados os controladores SDN e suas principais características. Na quarta seção é apresentado o serviço de nomes de domínio (DNS). Na quinta seção são apresentados os Middleboxes e suas características. Por fim, na última seção é apresentado o conceito de NFV *(Network Functions Virtualization)*.

2.1 Redes Definidas por Software - SDN

Redes Definidas por Software (SDN) são uma arquitetura emergente onde o controle da rede é desacoplado do encaminhamento de dados, permitindo que os dispositivos de rede sejam programáveis. O controle da rede, antes atrelado a cada dispositivo, é migrado para dispositivos de computação capazes de controlar vários elementos da rede, por exemplo switches. Assim, o hardware utilizado é abstraído. O controlador é implementado em outro dispositivo na rede, por exemplo um ou mais servidores, permitindo uma rede com controle logicamente centralizado [Open Networking Foundation, 2012].

A motivação para criação do paradigma SDN veio devido às várias dificuldades encontradas para inovar em redes de computadores, caracterizadas como complexas e difíceis de gerenciar [Feamster et al., 2013]. Os fatores que causam essas dificuldades são: a quantidade de protocolos utilizados; dificuldade de realizar testes em ambientes de produção. Essas várias dificuldades fazem com que autores caracterizem as redes de computadores que não utilizam o paradigma SDN como entidades calcificadas ou ossificadas [McKeown et al., 2008].

Redes Definidas por Software (SDN) estão mudando a forma de como projetamos e gerenciamos redes de computadores. Ao separar o plano de controle, é possível programar o plano de dados de vários elementos de rede via uma API *(Application Programming Interface)*. Um exemplo de API para SDN é o protocolo OpenFlow, explicitado na sessão 2.2. Ao permitir programação via interface padronizada, tarefas mais complexas podem ser desenvolvidas e automatizadas de maneira mais simples. Problemas de padronização também são eliminados, pois os hardwares de diferentes fabricantes passam a utilizar o mesmo padrão.

Antes de SDN introduzir a programação do plano de dados já existiam outras formas de permitir uma maior programabilidade para redes de computadores. O artigo "The Road to SDN" [Feamster et al., 2013] classifica em três eras responsáveis em trazer programação para redes de computadores, até alcançar o paradigma SDN. A primeira era aconteceu no início dos anos 90, e possui como representante a iniciativa chamada de Active Networking. Active Networking consistia em prover uma API para redes que permitia consultar os recursos disponíveis (processamento, armazenamento, fila de pacotes) em nós individuais.

A segunda era aconteceu durante o início dos anos 2000, e foi responsável pela separação entre os planos de controle e dados. O que motivou a separação foi a dificuldade de depurar e criar regras mais complexas para controlar switches e roteadores. Uma das contribuições relevantes é o sistema Ethane [Casado et al., 2007]. Ethane consiste em uma API para controle de acesso em uma rede empresarial. Desenvolvido na universidade de Stanford, seu projeto influenciou o desenvolvimento da API do protocolo OpenFlow.

A terceira era aconteceu em meados dos anos 2000. Esta época foi responsável pelo aparecimento do protocolo OpenFlow [McKeown et al., 2008] e os sistemas operacionais de rede, também conhecidos como controladores. Pesquisadores e agências de fomento tiveram interesse na ideia de experimentações em redes, encorajados pelo sucesso de projetos de experimentação de infraestrutura (PlanetLab [Bavier et al., 2004], Emulab [White et al., 2002]). Além de investimentos do governo para empreendimentos em larga escala foi possível a criação de novas soluções. Projetos que ganharam destaque oriundos desse entusiasmo foram o GENI (Global Environment for Networking Innovations) [1] e o projeto Europeu EU-FIRE (Future Internet Research and Experimentation) [2].

Enquanto isso, um grupo de pesquisadores de Stanford criou um projeto focado também em experimentação, só que em outro ambiente, as redes universitárias [Mc-

[1] https://geni.net
[2] https://www.ict-fire.eu/

Keown et al., 2008]. Este projeto seria o OpenFlow, uma API que implementa o paradigma SDN. Junto da criação do OpenFlow surgiu também o NOX [Gude et al., 2008], controlador SDN para o protocolo OpenFlow. Com a criação do NOX foi possível criar novas aplicações de controle de rede [Feamster et al., 2013]. O protocolo OpenFlow e seus controladores serão abordados nas sessões 2.2 e 2.3 respectivamente.

2.1.1 Arquitetura SDN

A arquitetura SDN pode ser representada de maneira simples por três elementos. A Figura 2.1 apresenta estes elementos. O número 1 representa a Camada de Infraestrutura, composta pelos dispositivos de rede, por exemplo switches, roteadores, pontos de acesso. O número 2 representa interface entre o plano de controle e o plano de dados, separados fisicamente.

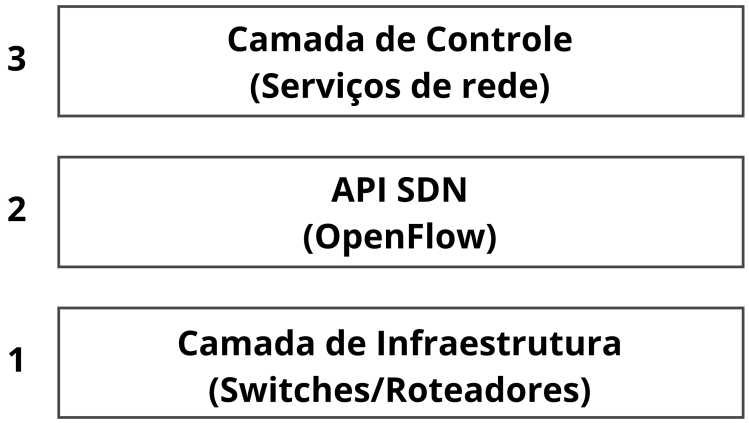


Figura 2.1. Arquitetura SDN

Esta interface pode ser representada por um protocolo, ou API, por exemplo o protocolo OpenFlow, que será apresentado na sessão 2.2. O número 3 representa a camada de controle, representada pelo software de controle SDN. Na camada de controle são implementados os serviços de rede, como switches de aprendizados nível 2, roteadores, Firewalls, serviços de DHCP entre outros. Estas são algumas das possíveis aplicações a partir da arquitetura SDN [Open Networking Foundation, 2012].

SDN proporciona uma série de benefícios para as redes de computadores que podem ser implementada em empresas ou operadoras. Entre eles podemos listar:

- Controle centralizado da rede e independente do fabricante.

- Melhora da automação e do gerenciamento da rede através de uma API para abstrair o hardware utilizado.
- Rápida capacidade de inovação para incluir novos serviços sem a necessidade de configurar individualmente os dispositivos ou esperar por atualizações dos fabricantes.
- Programabilidade independente do software e hardware utilizado, utilizando ambientes comuns de programação.
- Aumento da segurança e confiabilidade da rede por meio do gerenciamento automatizado e centralizado dos dispositivos de rede.
- Controle da rede com uma maior granularidade

2.1.2 Projetos que utilizam SDN

SDN vem sendo aplicada em vários projetos. Empresas consagradas como Google [Jain et al., 2013] e Facebook [3] [4] [5], já demonstraram interesse na tecnologia e implementaram soluções baseadas em SDN. Um curso de redes utilizando o Mininet [6] [Handigol et al., 2012], um emulador SDN, foi apresentado no SIGCOMM do ano de 2014 [7]. Scott Shenker, um dos idealizadores do protocolo OpenFlow, em uma palestra advoga as vantagens de SDN e faz um paralelo com outras áreas da computação em relação da dificuldade de inovar em redes de computadores [8]. Atualmente, a ONF (Open Network Foundation) [9] é responsável por discutir SDN e padronizar versões do protocolo OpenFlow.

2.2 O Protocolo OpenFlow

OpenFlow é a primeira interface de comunicação padronizada entre o controlador e a camada de infraestrutura de uma arquitetura SDN. O OpenFlow permite acesso direto e a manipulação do plano de encaminhamento dos dispositivos de rede, como switches e roteadores, físicos ou virtuais. A tecnologia OpenFlow baseada em SDN permite

[3] https://gigaom.com/2014/11/14/facebook-shows-the-promise-of-sdn-with-new-networking-tech/
[4] http://www.techcentral.ie/software-defined-networking/
[5] https://code.facebook.com/posts/717010588413497/introducing-6-pack-the-first-open-hardware-modular-switch/
[6] http://mininet.org
[7] http://conferences.sigcomm.org/sigcomm/2014/tutorial-mininet.php no sigcomm 2014
[8] https://www.youtube.com/watch?v=YHeyuD89n1Y
[9] https://onf.org

tratar a alta vazão das atuais aplicações, adaptar a rede para as constantes mudanças de negócio, e reduzir significantemente as operações e a complexidade de gerência [10].

O protocolo OpenFlow [McKeown et al., 2008] foi criado em 2008. Sua origem foi na faculdade de Stanford e teve como principal aplicação a rede do campus universitário. O projeto é baseada na implementação do sistema Ethane [Casado et al., 2007] autoria de Martín Casado. O design do switch utilizado no Ethane foi a base para a construção da API original do OpenFlow [Feamster et al., 2013].

2.2.1 Arquitetura OpenFlow

O switch OpenFlow, assim como a maior parte dos switches Ethernet e roteadores, utiliza tabelas de fluxos tipicamente construídas utilizando memórias do tipo TCAM. TCAM é um tipo de memória que permite endereçamento conteúdo por endereço. Este tipo de memória opera a taxas capazes de implementar serviços de Firewall, NAT, QoS e realizar a coleta de estatísticas de fluxo. Apesar de cada fabricante implementar a sua própria tabela de fluxos, o OpenFlow explora as características em comum que existem entre vários switches/roteadores para criar sua própria arquitetura.

Um switch OpenFlow é composto por pelo menos três partes. A primeira é a tabela de fluxos, com uma ação associada à cada entrada de fluxo para indicar como processar o fluxo. A segunda parte é o canal seguro *(Secure Channel)*, que conecta o switch ao controlador, permitindo que comandos e pacotes sejam trocados entre as entidades participantes. A terceira parte que compõe o switch é o protocolo OpenFlow, utilizado para realizar a comunicação. O protocolo OpenFlow provê uma forma padronizada de estabelecer a comunicação com um switch [McKeown et al., 2008].

Os switches OpenFlow podem ser caracterizados de duas formas. Os switches OpenFlow dedicados, estes são switches que não possuem processamento para as camadas 2 e 3, estes switches operam exclusivamente utilizando o protocolo OpenFlow. A outra categoria disponível são os switches Ethernet e roteadores comerciais de propósito geral. Estes switches possuem o protocolo OpenFlow adicionado como uma nova funcionalidade. Desta forma, não precisam utilizar a tecnologia OpenFlow obrigatoriamente.

Os switches OpenFlow dedicados possuem um plano de dados *"burro"*. O plano de dados é responsável por encaminhar os pacotes para as portas de destino, definidas por um controlador remoto. Os fluxos são definidos pelos campos existentes na tabela de fluxos implementada (podendo variar em cada versão do protocolo). Por exemplo,

[10] https://www.opennetworking.org/sdn-resources/openflow

um fluxo pode ser identificado por uma conexão TCP, ou todos pacotes oriundos de um certo endereço MAC, IP, pacotes de uma VLAN, ou todos os pacotes de uma porta do switch. Por exemplo, para experimentos que não utilizam pacotes IPv4, um fluxo pode ser definido como todos os pacotes que casam um cabeçalho específico.

Cada entrada de fluxo possui uma ação associada. As três ações obrigatórias que cada switch OpenFlow deve implementar são:

- **Encaminhamento dos pacotes do fluxo para a(s) porta(s) associada(s).** Esta ação permite que os pacotes sejam encaminhados pela rede.

- **Encapsulamento e encaminhamento dos pacotes do fluxo para o controlador.** Os pacotes são entregues para o canal seguro (*Secure Channel*), onde eles são encapsulados e enviados para o controlador. Tipicamente utilizado para o primeiro pacote de um fluxo novo, para que então o controlador possa decidir se o fluxo deve ser adicionado ou não à tabela de fluxos. Em experimentos, esta ação pode ser utilizada para encaminhar todos os pacotes ao controlador para processamento.

- **Descarte (*Drop*).** Pode ser utilizado em casos de segurança. Por exemplo, para evitar ataques de negação de serviço. Outra forma de utilização da ação *Drop* é para reduzir o excesso mensagens do tipo difusão (*broadcast*), utilizadas para descobrir serviços na rede.

Outra ação importante, porém não obrigatória, é a **reescrita dos cabeçalhos dos pacotes**. O controlador pode inserir regras para que pacotes sejam enviados ao controlador e tenham seus cabeçalhos alterados. Por exemplo, os campos que podem ser alterados são: Endereços IP e MAC tanto de origem como de destino.

Uma entrada na tabela de fluxos possui três campos. O primeiro campo é composto pelos cabeçalhos dos pacotes que identificam o fluxo (campos de casamento). O segundo campo é composto pelos dados estatísticos do fluxo (contadores estatísticos). São armazenados pelos contadores estatísticos a quantidade de bytes, quantidade de pacotes, e o tempo que o último pacote casou com aquele fluxo. Por último, o terceiro campo, que é composto pelas ações que definem como os pacotes devem ser processados. A figura 2.2.1 representa a tabela de fluxos de um switch OpenFlow versão 1.0.

Os cabeçalhos são extraídos dos protocolos das camadas L2/L3/L4 e são utilizados para realizar o casamento das regras de fluxo. Os contadores salvam estatísticas para o fluxo, como a quantidade de bytes ou pacotes que casaram com aquele fluxo.

Por último, as ações são responsáveis por informar como o fluxo irá se comportar em caso de casamento.

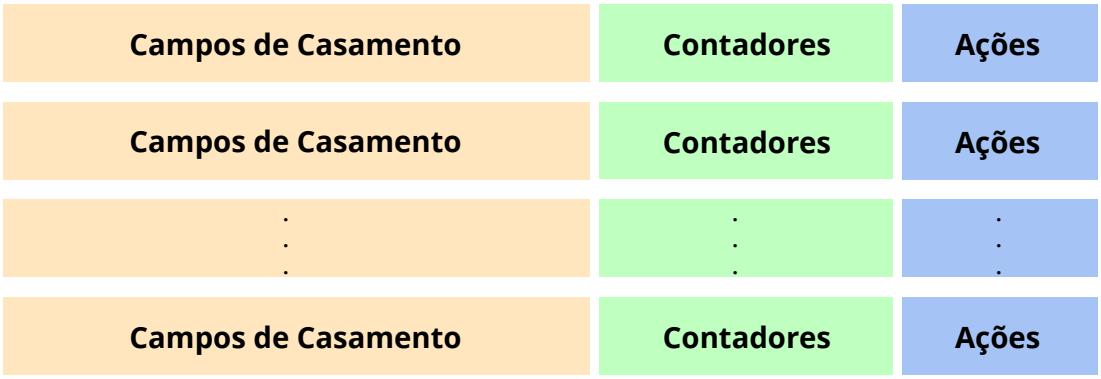


Figura 2.2. Tabela de Fluxos OpenFlow

2.2.2 Benefícios da Arquitetura OpenFlow

As vantagens que empresas e provedores de internet *(ISPs)* podem proporcionar por meio de uma arquitetura SDN utilizando OpenFlow são [Open Networking Foundation, 2012]:

Controle centralizado do ambiente: O controlador SDN pode controlar qualquer dispositivo que utilize a mesma API SDN, por exemplo OpenFlow, independente do fabricante. Dentre os dispositivos possíveis, pode-se listar switches, roteadores, e switches virtuais. Ao invés de gerenciar uma série de grupos de dispositivos de diferentes fabricantes, o administrador de redes pode utilizar ferramentas que utilizam SDN para implementar, configurar, e atualizar serviços em toda a rede.

Redução da complexidade através da automação: Switches SDN que utilizam o protocolo OpenFlow oferecem um *framework* flexível para gerenciamento e automação. Assim, tornando possível desenvolver ferramentas para automatizar várias tarefas de gerenciamento que são realizadas de forma manual. Estas ferramentas de automação irão reduzir o custo adicional (*overhead*), e diminuir a instabilidade da rede gerada por possíveis erros dos operadores.

Maior grau de inovação: A adoção do paradigma SDN acelera a inovação permitindo que administradores de rede literalmente programem e reprogramem a rede em tempo real. Assim, correspondendo com as necessidades de negócio e os novos requisitos de usuário. Por exemplo, ao virtualizar a infraestrutura de rede e abstraí-la

de serviços de rede individuais, SDN e OpenFlow dão aos administradores de rede a habilidade de moldar o comportamento da rede. Desta forma, introduzir novos serviços e novas capacidades à rede em questão de horas.

Aumento da segurança da rede: SDN torna possível para administradores de rede declarar políticas em alto nível, as quais são instaladas na infraestrutura via protocolo OpenFlow. Uma arquitetura SDN que utiliza o protocolo OpenFlow elimina a necessidade de configurar individualmente cada dispositivo da rede toda vez que um ponto, serviço ou aplicação é adicionada, movida, ou quando uma política for alterada, o que diminui a chance de erros devido à falhas de configuração ou políticas inconsistentes.

Maior controle da rede: O controle de fluxo utilizando OpenFlow permite que administradores de redes apliquem políticas mais detalhadas, com maior granularidade de uma forma abstraída e automatizada. Este controle permite que operadores de arquiteturas em nuvem suportem multi-inquilinos enquanto o tráfego é isolado e seguro.

Melhor experiência de usuário: Ao centralizar o controle da rede e fazer o estado da informação disponível para aplicações de alto nível de abstração, uma infraestrutura SDN pode se adaptar melhor às necessidades dinâmicas do usuário. Por exemplo, um provedor pode introduzir um serviço de vídeo que provê para os usuários de um plano *premium* a resolução mais alta de uma forma automatizada e transparente. Hoje, usuários devem explicitamente selecionar a opção de resolução, a qual a rede pode ou não suportar, resultando em atrasos e interrupções que degradam a experiência de usuário. Com uma arquitetura SDN utilizando Openflow, a aplicação seria capaz de detectar a largura de banda disponível na rede em tempo real e automaticamente ajustar a resolução do vídeo de acordo com a banda disponível.

2.2.3 Limitações do Protocolo OpenFlow

Apesar de todo o seu poder o protocolo OpenFlow possui limitações. O OpenFlow ainda não oferece suporte para para inspeção da carga útil do pacote (*Deep Packet Inspection-DPI*) ou remontagem da conexão (*connection reassembly*). Assim, o OpenFlow sozinho não é capaz de implementar eficientemente funcionalidades sofisticadas de *Middleboxes* [Feamster et al., 2013].

Outro problema relacionado ao OpenFlow é o número excessivo de pacotes que são enviados para o controlador [Mogul et al., 2010]. Assim, prejudicando o desempenho das aplicações. Uma forma de evitar este comportamento é carregando as regras proativamente. Esta abordagem permite que a quantidade de pacotes enviados ao controlador seja menor, possibilitando um desempenho melhor da rede.

2.2.4 Versões do Protocolo OpenFlow

Atualmente o protocolo OpenFlow está na versão 1.5. A tabela 2.1 apresenta todas as versões já criadas para o protocolo OpenFlow e algumas de suas características. A segunda e a terceira coluna da tabela apresentam os campos para casamento obrigatórios e opcionais para cada versão do protocolo. A quarta e a quinta coluna apresentam a quantidade de ações obrigatórias e opcionais. A última coluna apresenta o lançamento da especificação adotada para cada versão. Essa tabela foi construída utilizando as especificações oficiais para cada versão do protocolo OpenFlow. As especificações estão disponíveis no site da ONF [11].

Tabela 2.1. Versões do Protocolo OpenFlow. Baseada na tabela de [Kreutz et al., 2015]. A versão 1.0 do OpenFlow considera a ação de encaminhamento do switch como uma ação virtual, por isso são contabilizadas apenas duas ações. Nas próximas versões ela já é considerada como uma ação obrigatória.

Versão	Casamento Ob.	Casamento Op.	Ações Ob.	Ações Op.	Lançamento
1.0	12	-	2	3	12/2009
1.1	15	-	3	4	02/2011
1.2	13	23	3	4	12/2011
1.3	13	27	3	4	06/2012
1.4	13	29	3	4	10/2013
1.5	14	31	3	6	12/2014

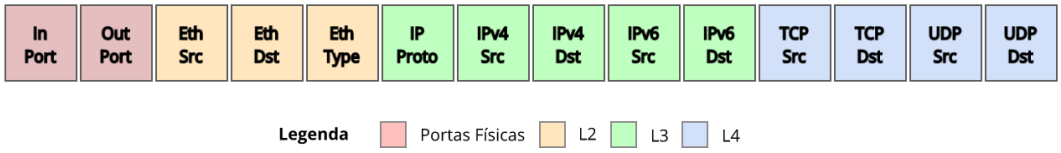
A figura 2.3 apresenta os campos obrigatórios de casamento para versão 1.5 do protocolo OpenFlow. No total são 14 campos de cabeçalhos obrigatórios para casamento e identificação dos fluxos. Os campos nas cores vermelha identificam as portas físicas de entrada e saída do switch. Os campos na cor amarela são respectivos à camada 2, enlace, protocolo Ethernet. Os campos em verde são correspondentes aos protocolos da camada 3, IPv4 e IPv6. Por último, em azul, os campos da camada 4, de transporte dos protocolos TCP e UDP. A versão 1.0 do OpenFlow não possui apenas 3 desses campos obrigatórios. São eles a porta física de saída e os campos IPv6 de origem e destino. O IPv6 foi introduzido ao OpenFlow a partir da versão 1.2.

2.2.5 Implementações em Hardware e Software do protocolo OpenFlow

Vários fabricantes adotaram o protocolo OpenFlow. Entre eles podem ser citados Juniper, HP, Huawei, Brocade, Pica8 [Kreutz et al., 2015]. Por exemplo, um switch Pica8

[11] https://www.opennetworking.org/ja/sdn-resources-ja/onf-specifications/openflow

Figura 2.3. OpenFlow 1.5 Cabeçalhos Obrigatórios



Gigabit com 48 portas, executando a versão 1.4 do protocolo OpenFlow custa cerca de U$4.000,00 dólares [12]. Enquanto um switch Cisco Gigabit, sem o protocolo OpenFlow, também com 48 portas custa cerca de U$2.000,00 dólares [13]. Outros switches OpenFlow e seus fabricantes podem ser encontrados na página da ONF [14].

Outra forma possível de utilizar a tecnologia OpenFlow é por meio de switches em software. Esses switches podem ser executados em hardware genéricos. Por exemplo, PCs comuns arquitetura x86 ou pontos de acesso Wi-Fi. Os switches virtualizados também pode ser utilizados para controle de VMs. O exemplo mais comum é o Open vSwitch (primeira linha da tabela 2.2), incluído no kernel do Linux desde a versão 3.10. O Open vSwitch implementa até a versão 1.5 do protocolo OpenFlow. O Open vSwitch também é utilizado pela VMware para fazer o chaveamento de pacotes em seus programas de virtualização [15]. Open vSwitch [16] é escrito na linguagem de programação C e é aplicado em produção/desenvolvimento possuindo funcionamento em nível de kernel do sistema operacional.

Tabela 2.2. Versões Open Source do Switch OpenFlow. Tabela baseada em [Goransson & Black, 2014]

Switch	Organização	Versão OF	Linguagem	Foco
Open vSwitch	Nicira	1.0 até 1.5	C	Desenvolvimento
OpenFlow Reference	Stanford	1.0	C	Pesquisa
CPqD	CPqD	1.3	C	Pesquisa
Pantou	Stanford	1.0	C	Desenvolvimento
Indigo	Big Switch	1.3	C	Desenvolvimento
LINC	Infoblox	1.2, 1.3 e 1.4	Erlang	Desenvolvimento

[12] http://store.netgate.com/Pica8/P-3297.aspx
[13] https://www.amazon.com/dp/B003ICX9PK?psc=1
[14] https://www.opennetworking.org/sdn-openflow-products?start=20
[15] https://github.com/openvswitch/ovs/blob/master/FAQ.md
[16] http://openvswitch.org/

A tabela 2.2 apresenta outras versões em software do protocolo OpenFlow. A segunda linha representa o switch criado pela universidade de Stanford. O OpenFlow Reference Switch [17] [18]. Este switch implementa as funcionalidades básicas do protocolo OpenFlow e possui como foco a pesquisa. Já que o seu foco é pesquisa, ele é executado em espaço de usuário, o que facilita a sua depuração e implementação. Contudo, o desempenho é bem abaixo em comparação com outros switches que são executados em espaço de kernel. Cada pacote que passa pelo switch precisa ser copiado do espaço de kernel para o espaço de usuário, o que compromete o seu desempenho.

O protocolo OpenFlow versão 1.3 [19], representado pela terceira linha da tabela 2.2, apresenta o switch criado pelo CPqD. O CPqD é uma instituição de pesquisa brasileira. O switch desenvolvido foi ganhador de um prêmio de U$50.000,00 dólares [20] em um concurso de pesquisa organizado pela Open Network Foundation. O switch possuiu o principal objetivo de lançar a versão 1.3 do protocolo OpenFlow, incluindo novos campos de cabeçalho para casamento.

O switch Pantou é designado para operar em pontos de acesso (*access points*) Wi-Fi. Seu desenvolvimento é realizado pela universidade de Stanford e escrito na linguagem C. Pantou foi escrito em cima do sistema OpenWRT e é baseado no switch OpenFlow Reference de Stanford. Mais informações podem ser adquiridas no site do projeto [21].

O switch Indigo [22] foi desenvolvido pela empresa Big Switch, e o seu foco é desempenho. Seu código é escrito em C e pode ser também utilizado em switches dedicados. Por último, o switch LINC [23]. LINC foi criado para ser utilizado em computadores comuns (*commodity*) de arquitetura x86 e pode ser executado em vários sistemas operacionais como Linux, Solaris, Windows, MacOS, e FreeBSD, onde o ambiente de execução Erlang esteja disponível.

[17] http://archive.openflow.org/wp/downloads/
[18] https://github.com/mininet/openflow
[19] https://github.com/CPqD/ofsoftswitch13
[20] https://www.opennetworking.org/member-login/26-news-and-events/press-releases/1431-open-networking-foundation-announces-openflow-driver-contest-winner
[21] http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT
[22] http://www.projectfloodlight.org/indigo/
[23] https://github.com/FlowForwarding/LINC-Switch

2.3 Controladores SDN - Sistemas Operacionais de Redes

Controladores SDN são o cérebro da rede em uma arquitetura SDN, são responsáveis por executar as aplicações que atuam como controle de uma rede SDN. Controladores SDN gerenciam os fluxos para os switches/roteadores. A comunicação entre os dispositivos é realizada via API padronizada *(Southbound API)*, um exemplo de API é a OpenFlow. O controlador também pode disponibilizar outra API para se comunicar com aplicações da camada de negócios *(Northbound API)*, e assim construir programas capazes de extrair informações da rede [24].

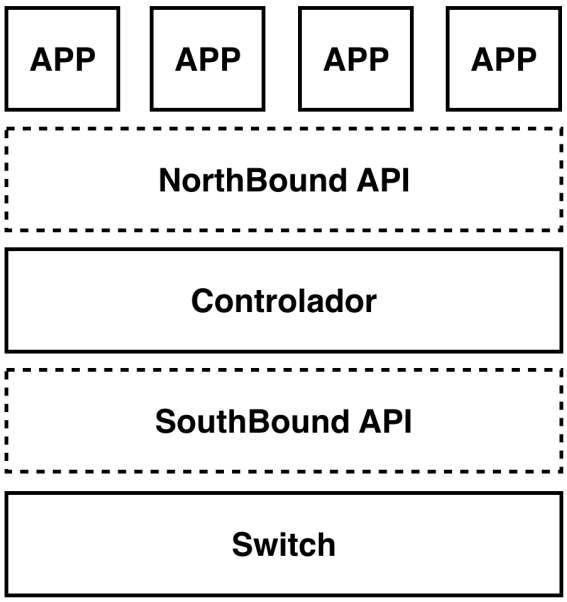


Figura 2.4. Arquitetura SDN e as *SouthBound* e *NorthBound* APIs

A figura 2.4 representa uma arquitetura SDN e as *NorthBound* e *SouthBound* APIs. No primeiro nível é identificado o switch/roteador. O segundo nível representa a API de comunicação com o controlador *SouthBound API*. No terceiro nível o é representado o controlador. O quarto nível é representado pela API de comunicação entre o controlador e as aplicações. No último nível são apresentadas as aplicações possíveis de serem construídas utilizando a *Northbound API*.

[24]https://www.sdxcentral.com/sdn/definitions/sdn-controllers/

2.3.1 Benefícios Controladores SDN

A configuração dos dispositivos de rede hoje em dia é totalmente atrelada ao fabricante. Assim, cada alteração que precisa ser realizada depende da interface de configuração e das características do dispositivo. Desta forma, é difícil transferir configurações de um dispositivo para outro, seja ele do mesmo fabricante, mas modelos diferentes ou de fabricantes diferentes.

Esse modo de configuração se assemelha ao modo de programação utilizado antes da criação de sistemas operacionais. Cada máquina era programada de acordo com o seu conjunto de instruções. Isso tornava difícil de portar os códigos para outra arquitetura e dificultava a sua programação. Para isso foram criados os sistemas operacionais. Uma camada de abstração para permitir que programas possam ser criados, depurados, testados, implementados de forma mais fácil.

Inspirado nisso, foi identificada a necessidade de criação de um sistema operacional para redes. Um sistema que crie uma camada de abstração, simplificando a forma que são controladas as redes de computadores e permitindo um controle logicamente centralizado de toda a arquitetura. Não é papel do sistema controlar a rede, e sim das aplicações que serão construídas utilizando a sua camada de abstração (por exemplo a API OpenFlow).

Assim foram concebidos os sistemas operacionais de rede, também conhecidos como controladores SDN. O controlador adiciona e remove entradas de fluxos nas tabelas do switch/roteador. Em arquiteturas SDN o controle é desacoplado do plano de dados. Esta característica permite criar programas para controlar a rede, permitindo uma arquitetura programável, automatizada e de fácil gerência [Gude et al., 2008].

2.3.2 Controladores OpenFlow

O primeiro controlador OpenFlow foi o NOX [Gude et al., 2008] criado pela Nicira em 2008 junto com o protocolo OpenFlow. Após isso, a Nicira foi vendida para a VMware e o NOX teve o seu código aberto e serviu como inspiração para criação de novos controladores [25]. A tabela 2.3 apresenta outros controladores populares, a versão do protocolo OpenFlow suportada, qual a linguagem que utilizam para programar e o foco, se são utilizados em ambientes de pesquisa ou desenvolvimento.

O controlador POX [26] apresentado na tabela 2.3 é a versão em Python do controlador NOX. O controlador POX possui foco em pesquisas. A linguagem de programa-

[25] https://www.sdxcentral.com/sdn/definitions/sdn-controllers/
[26] https://github.com/noxrepo/pox

ção utilizada pelo POX é a Python. O controlador NOX por outro lado utiliza C++, permitindo um desempenho melhor para as aplicações.

POX e NOX são controladores orientados a eventos. A cada evento pré-definido pelo protocolo OpenFlow, é possível que o administrador de redes escreva as rotinas desejadas. Alguns exemplos de eventos são, estabelecimento de conexão entre controlador e switch, entrada de pacote no switch, entre outros.

Tabela 2.3. Controladores OpenFlow *Open Source*. Tabela baseada em [Goransson & Black, 2014]

Controlador	Organização	Versão OF	Linguagem	Foco
NOX	ICSI	1.0	C++	Desenvolvimento
POX	ICSI	1.0	Python	Pesquisa
RYU	NTT	1.0 a 1.5	Python	Desenvolvimento
ONOS	On.Lab	1.0 e 1.3	Java	Desenvolvimento
Floodlight	Big Switch	1.0 a 1.4	Java	Desenvolvimento
OpenDaylight	OpenDaylight	1.0 a 1.3	Java	Desenvolvimento

O controlador Ryu é desenvolvido em Python e seu código é aberto utilizando licença Apache 2.0. As versões OpenFlow 1.0 até a 1.5 são suportadas pelo Ryu. O ponto forte do controlador Ryu é fornecer aos administradores de rede uma API (Northbound API) para criação de aplicações de gerência e controle. O nome Ryu se origina da palavra em japonês fluxo *(flow)*.

O controlador ONOS [27] [Berde et al., 2014] possui foco em desempenho, escalabilidade, e disponibilidade. O seu código é escrito na linguagem de programação JAVA. As versões do protocolo OpenFlow suportadas pelo ONOS são 1.0 e versão 1.3. Outro controlador que utiliza a linguagem Java é o Floodlight [28]. Desenvolvido pela organização Big Switch o Floodlight possui como características o fácil uso e a ampla adoção por switches físicos e virtuais. O Floodlight suporta as versões 1.0 a 1.4 do OpenFlow. O último controlador da tabela 2.3 é o OpenDayLight [29] também utiliza Java como linguagem de programação, permitindo seu funcionamento em várias plataformas. As versões do OpenFlow suportadas pelo OpenDayLight são a 1.0 e 1.3. O ponto forte do OpenDayLight é a adoção de uma arquitetura baseada em microsserviços.

Além dos controladores *Open Source* citados na tabela 2.3 outro controlador que se destaca é o Onix [Koponen et al., 2010]. Após a aquisição da Nicira pela VMware, a Nicira passou a desenvolver o Onix em parceria com a NTT e Google. O ponto forte

[26] http://osrg.github.io/ryu/
[27] http://onosproject.org/
[28] http://www.projectfloodlight.org/floodlight/
[29] https://www.opendaylight.org/

do Onix é o seu aspecto escalável e voltado para grandes sistemas em produção. Uma versão modificada do Onix é implementada na arquitetura WAN da Google [Jain et al., 2013].

2.4 DNS - *(Domain Name System)*

A quantidade de computadores que estavam conectados à Internet (ARPANET) durante a década de 70 era limitado. Contudo, acessar os computadores participantes utilizando endereços de rede não é uma tarefa simples para humanos. Então, foram atribuídos nomes às máquinas para facilitar o acesso, evitando o uso do endereço de rede. Os nomes dos computadores/servidores participantes da Internet naquela época eram armazenados no arquivo `"/etc/hosts"` em distribuições UNIX de sistemas operacionais. Hoje este arquivo ainda é utilizado para armazenar os nomes das máquinas. Contudo, com o avanço da tecnologia e com o aumento de participantes na Internet esse tipo de armazenamento de nomes se tornou ineficaz. A dificuldade consiste em manter o arquivo `"/etc/hosts"` atualizado, de acordo com as dos nomes das máquinas e seus endereços.

A dificuldade de armazenar e manter atualizada a relação entre o nome da máquina e endereço criou a necessidade de criar uma nova solução. Para isso foi criado o DNS. DNS é o acrônimo para *Domain Name System*, em português Sistema de Nomes de Domínio. O DNS foi criado por Paul Mockapertis integrante do Instituto de Sistemas de Informação da Universidade do Sul da Califórnia (USC-EUA) no ano de 1984. As RFCs que definiram o protocolo DNS foram 882, 883, 1034 e em 1987 foi definida a última e atual RFC, a 1035 [30].

O DNS é uma base de dados distribuída. De forma simples, o DNS funciona respondendo requisições sobre nomes de domínio com os endereços IP que ele possui armazenado em sua base de dados. Os endereços dos servidores DNS são salvos nas configurações de rede dos usuários. Assim, quando o usuário requisita um nome de domínio, esta consulta é encaminhada para o servidor DNS. O cabeçalho de um pacote DNS é demonstrado na figura 2.5.

A primeira linha representa o identificador do pacote. A segunda linha possui os metadados sobre o pacote DNS. A terceira linha armazena a quantidade de questões que existem nesse pacote. A quarta linha apresenta a quantidade de respostas que o pacote DNS possui. A quinta linha apresenta a quantidade de registros para servidores de

[30] https://www.ietf.org/rfc/rfc1035.txt

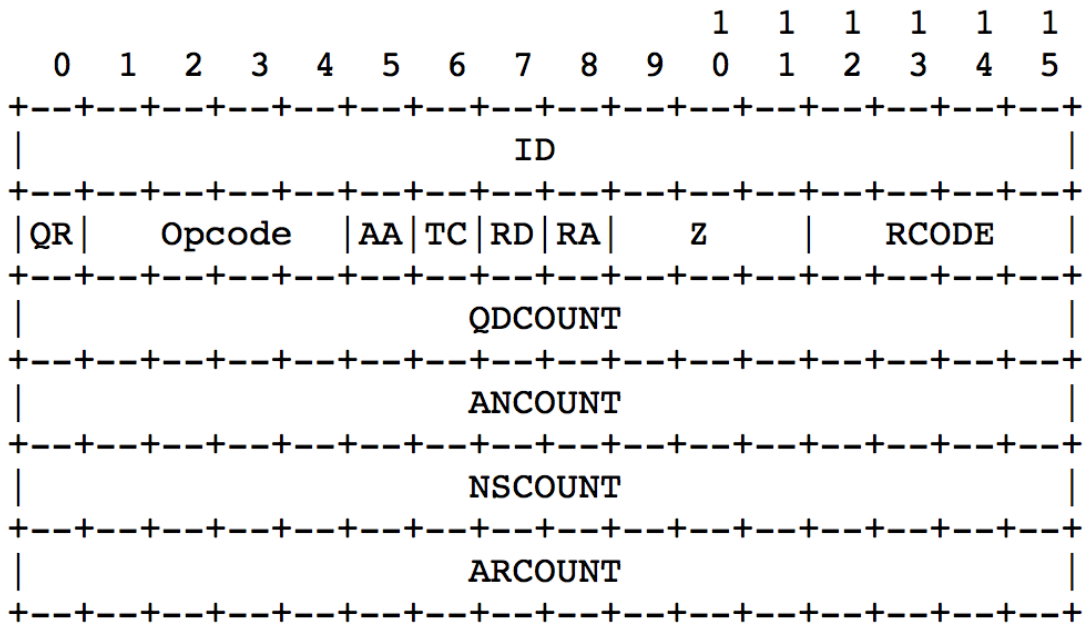


Figura 2.5. Cabeçalho do pacote DNS

nomes obtidos. Por último, a sexta linha apresenta a quantidade de registros adicionais obtidos.

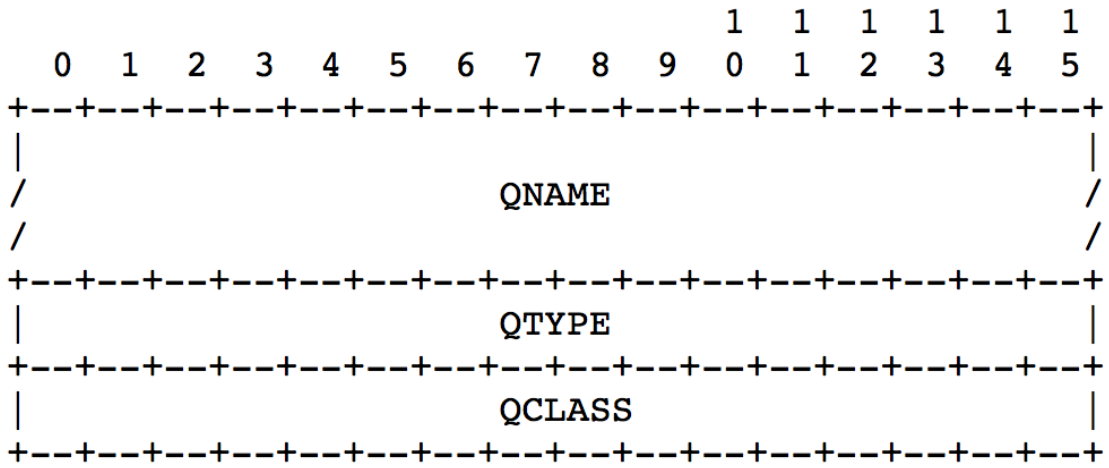


Figura 2.6. Estrutura de uma questão DNS

O pacote DNS que a realiza a consulta DNS é estruturado da seguinte forma representado pela figura 2.6. A primeira linha, representada pelo campo QNAME apresenta a o nome de domínio consultado pelo usuário. A segunda linha, QTYPE, representa o

tipo de consulta realizada, e por último, na última linha, QCLASS representa a classe da consulta realizada

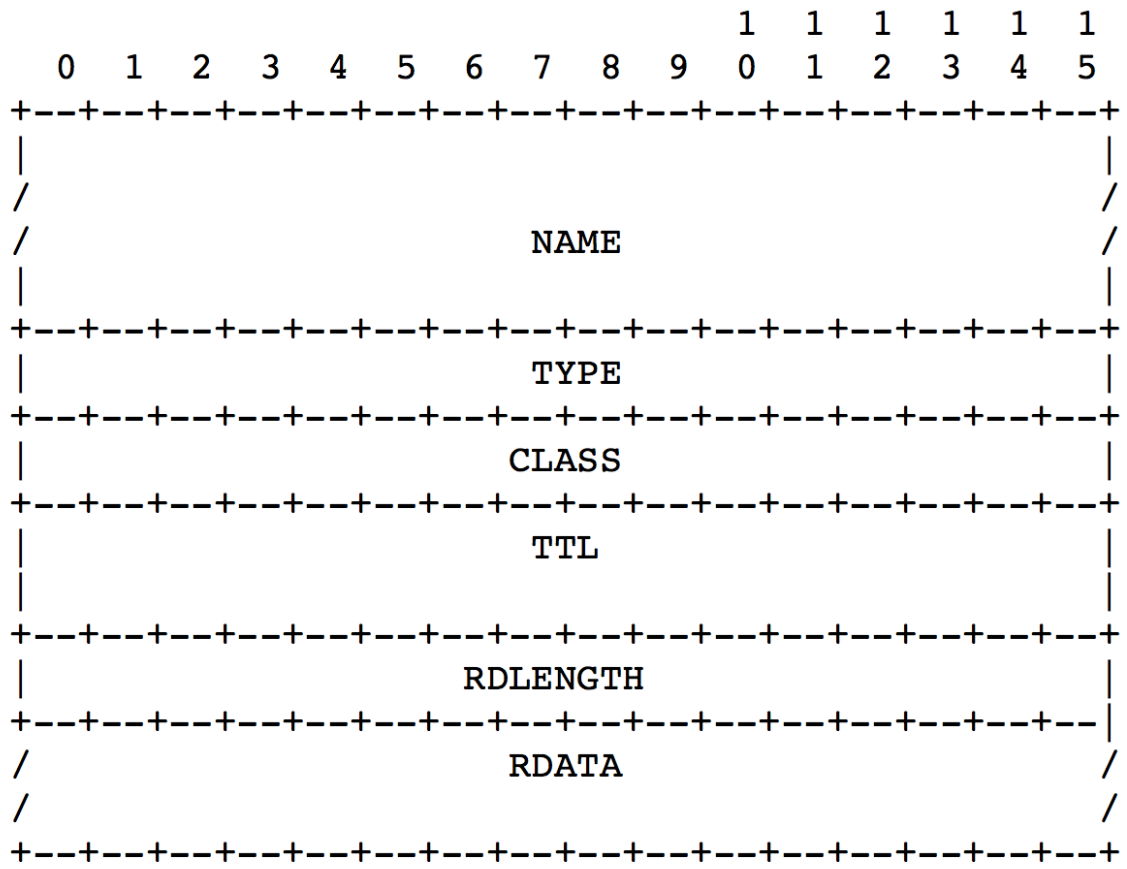


Figura 2.7. Estrutura de uma resposta DNS

A estrutura de um pacote DNS de resposta pode ser conferida na figura 2.7. O primeiro campo do pacote representa o nome retornado para a consulta realizada. Caso este noje já tenha sido referenciado em algum campo do pacote ele é preenchido com um ponteiro para o nome. O segundo campo da resposta DNS é o tipo da resposta obtida, exemplos de tipo de resposta são A (IPv4), AAAA (IPv6), CNAME (Canonical Name). O próximo campo para resposta DNS é o campo classe, alguns tipos presentes na classe são IN (Internet), CS, CH e HS. O campo TTL representa o Time To Live, isto é o tempo de vida para aquela informação em segundos. O campo RDLENGTH apresenta a quantidade de bytes contida no campo RDATA. Por fim, o campo RDATA possui as respostas para a consulta realizada, caso seja uma consulta do tipo A, este campo é composto por pelo menos um endereço IPv4.

Um servidor DNS atende requisições utilizando o protocolo UDP e a porta 53 [Mockapetris, 1987]. Uma das implementações de servidores mais utilizadas do

DNS é o `Bind` [Liu & Albitz, 1996]. Em uma distribuição Linux é possível consultar qual o endereço do servidor DNS é utilizado ao acessar o arquivo `/etc/resolv.conf`. Uma forma de fazer consultas DNS é via o utilitário `nslookup`.

2.5 Middleboxes

A palavra Middlebox foi cunhada por Lixia Zhang e foi designada para identificar um fenômeno na Internet. Um Middlebox é definido como um dispositivo intermediário que desempenha funções diferentes das funções de um roteador IP no caminho de um *host* de origem para um *host* de destino [31].

Um Middlebox pode ser uma máquina física à parte na rede, mas isso não exclui a possibilidade de um Middlebox ser uma entidade virtualizada. O Middlebox pode realizar várias tarefas sobre um fluxo de um pacote IP, mas ele nunca será o sistema fim de uma sessão de alguma aplicação. Os Middleboxes não são utilizados em protocolos abaixo da camada IP.

Alguns exemplos de aplicações implementados por Middleboxes são NAT *(Network Address Translator)*, Firewalls L3 (camada IP), Firewalls L7 (camada de aplicação), Proxies, Caches, Balanceadores de Carga *(Load Balancers)*, Sistemas de Prevenção e Detecção de Intrusos (IPS/IDS), entre outras aplicações. Estes Middleboxes podem ser implementados um dispositivo para propósito específico de arquitetura proprietária, como também podem ser implementados em Software. Neste caso, um computador genérico utilizando arquitetura x86, executaria uma aplicação funcionando como um Middlebox.

Middleboxes são plataformas caras e fechadas, com pouca ou nenhuma capacidade de extensão. Além disso, Middleboxes são adquiridos de fabricantes independentes e são implementados como dispositivos isolados com pequena coesão em como os Middleboxes são gerenciados. Como os requisitos de rede continuam crescendo em quantidade e variedade, Middleboxes são dispositivos que influenciam no crescimento dos custos de infraestrutura e de manutenção [Sekar et al., 2012; Sherry et al., 2012]. Em uma pesquisa apresentada no artigo: "The Middlebox Manifesto" [Sekar et al., 2011], apresenta o número de Middleboxes presentes em empresas. Ao somar todas aplicações de Middleboxes existentes, a quantidade de Middleboxes chega a representar cerca de 70% da quantidade de roteadores presentes em uma empresa. O que ratifica o quanto os Middleboxes são importantes. Vale lembrar que são tipos de dispositivos diferentes.

[31] https://tools.ietf.org/html/rfc3234

Em estudo focado em levantar as falhas que ocorrem em *Data Centers* [Gill et al., 2011] foi apresentado que 1 a cada 5 Middleboxes que executam *Load Balancers* apresentam falhas. Contudo, a maioria dos estudos para aprimorar os dispositivos de rede são focados na camada L2/L3. A camada de aplicação, área que os Middleboxes atuam ficam à margem. Outro ponto que reforça a importância dos Middleboxes, em específico aplicações voltadas para segurança, é a quantidade de dinheiro investida. Para o ano de 2010 foram estimados 6 bilhões de dólares gastos em aplicações para segurança de rede, e são estimados 10 bilhões para o ano de 2016 [Sekar et al., 2011].

Baseado nesses problemas foram propostas várias soluções para serem aplicadas aos Middleboxes. Uma das propostas é conhecida como *Middlebox Consolidation*. O termo em inglês *consolidation* nesse aspecto possui o sentido de colocar vários Middleboxes que possuem operações similares no mesmo hardware. Assim, aproveitar o pipeline de processamento do pacote. O ponto forte dessa aplicação é simplificar a arquitetura utilizando uma quantidade menor de Middleboxes, reduzir o custo, e aumentar o desempenho da rede [Sekar et al., 2012]. Uma proposta diferente é apresentada por [Sherry et al., 2012] no artigo intitulado "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service". A proposta desse artigo é desviar o tráfego para um Data Center terceirizado onde seria realizado o processamento relativo aos Middleboxes.

O artigo "Middlebox Manifesto" [Sekar et al., 2011] propõe aplicar princípios similares aos de SDN para o desenvolvimento de Middleboxes. São propostas três alternativas, a implementação de aplicações Middlebox que desacoplam o hardware do software, a de consolidação Middleboxes, representando a inclusão de várias aplicações Middlebox em um único hardware. Por último, é proposta a criação de uma API para permitir que Middleboxes possuam gerência logicamente centralizada. Assim, problemas como padronização, controle, gerência são mitigados. Contudo, surgem novos desafios quanto à nova arquitetura do Middlebox. Estes desafios são citados no artigo.

A integração entre Middleboxes e SDN é utilizada em uma técnica chamada de *traffic steering* [Bremler-Barr et al., 2014; Qazi et al., 2013]. O *traffic steering* consiste em classificar os fluxos a partir dos cabeçalhos dos pacotes das camadas L2 a L4 para direcioná-los para os Middleboxes corretos. Desta forma, evitando que o fluxo seja direcionado para aplicações que não precisam inspecioná-lo.

2.6 Network Functions Virtualization (NFV)

Network Functions Virtualization (NFV) [Chiosi et al., 2012] possui o foco de transformar a forma de como as redes são estruturadas e evoluir as tecnologias de TI. NFV possui como objetivo consolidar vários tipos de equipamentos em servidores, switches e *storages*. Estes dispositivos podem ser localizados em *Data Centers*, em nós na rede, e nos usuários finais. NFV envolve a implementação de aplicações de rede em software que possam ser executados em uma variedade de hardware/servidores. Assim, eles podem ser movidos, ou instanciados, em vários pontos na rede quando necessário, sem a necessidade de instalação de um novo equipamento.

NFV nasceu em Outubro de 2012 quando AT&T, BT, China Mobile, Deutsche Telekom e várias outras empresas de telecomunicação introduziram o *whitepaper* intitulado "NFV An Introduction, Benefits, Enablers, Challenges & Call for Action". Um novo comitê foi montado sob supervisão do ETSI *(European Telecommunications Standards Institute)*. Este comitê irá trabalhar para criar o padrão NFV. NFV pode implementar roteadores, Firewalls, Load Balancers, e outro dispositivos de rede, todos utilizando virtualização e executados em hardware de baixo custo [32].

O desenvolvimento de sistemas NFV são motivados por problemas gerados por Middleboxes. Com o passar dos anos Middleboxes se tornaram peças fundamentais em redes de computadores. Apesar da sua utilidade, Middleboxes possuem uma série de problemas. Alguns desses problemas são acarretados devido ao fato dos Middleboxes serem arquiteturas implementadas em hardware, de alto custo, difíceis de gerenciar, e com funcionalidades difíceis ou impossíveis de mudar.

NFV busca reduzir o custo de equipamentos, reduzir o consumo de energia através da consolidação de equipamentos e diminuir o tempo de lançamento de novas aplicações. As aplicações construídas utilizando software em ambientes virtualizados possuem o seu desenvolvimento facilitado, porque não estão atreladas a uma arquitetura fixa de hardware. Dividir a utilização de uma única plataforma (hardware) com várias outras aplicações, usuários ou inquilinos permite que o administrador de redes compartilhe recursos entre serviços e diferentes usuários.

Além disso NFV visa permitir que os serviços possam ser escalados *(scaled up/-down)* de acordo com a necessidade, permitir uma grande variedade de eco-sistemas, e encorajar a inovação e a criação de sistemas abertos. Por fim, atrair a experimentação, pesquisas acadêmicas e atrair novos serviços e fluxos de receita de forma rápida e com risco menor [Chiosi et al., 2012].

[32] http://wikibon.org/wiki/v/Network_Function_Virtualization_or_NFV_Explained

Para acelerar o desenvolvimento de aplicações para plataformas NFV. A Linux Foundation anunciou uma plataforma de referência para projetos abertos em NFV, a OPNFV [33]. A OPNFV vai promover uma rede *open source* que possui o objetivo de reunir as empresas para acelerar a inovação. OPNFV vai reunir provedor de serviços, provedores de nuvem e infraestrutura, desenvolvedores e clientes para criar uma plataforma aberta para acelerar o desenvolvimento e implementação de sistemas NFV [34]. Intel, IBM, DELL, VMware, AT&T, Cisco, Red Hat, HP, Juniper, Ericsson, Lenovo, Nokia, Huawei são algumas das empresas que participam da OPNFV.

[33]https://opnfv.org
[34]https://www.sdxcentral.com/nfv/definitions/opnfv/

Capítulo 3

Trabalhos Relacionados

Neste capítulo são apresentadas as propostas relacionadas ao trabalho apresentado nessa dissertação. As propostas estão separadas em sub-temas para agrupar e organizar a discussão. A primeira discussão será sobre temas que abordam a extensão do protocolo OpenFlow. Após isso, serão tratados os temas que apresentam novas propostas para projeto e arquitetura de rede. Em seguida, serão tratados os NFVs e Middleboxes, sistemas que implementam aplicações de rede. Por último, os já extintos switches de conteúdo. Ao fim deste capítulo é apresentada a tabela de comparação das propostas salientando pontos positivos e negativos de cada proposta.

3.1 Propostas de extensão do protocolo OpenFlow

A proposta desta dissertação é estender o protocolo OpenFlow para atender nomes de domínio. Atualmente, o padrão OpenFlow está na versão 1.5. A cada versão são adicionados mais campos à estrutura de casamento, mas ainda não é possível realizar o casamento de dados existentes no *payload* [Goransson & Black, 2014].

Existem algumas propostas para estender o OpenFlow. Um exemplo é SDN para MPLS [Davie & Farrel, 2008], com a adição de portas virtuais para suportar noções de encapsulamento e desencapsulamento empregados em MPLS.

Devo-Flow modifica o modelo OpenFlow. Ele muda a divisão de responsabilidades entre controladores e switches [Mogul et al., 2010]. Os autores afirmam que o OpenFlow possui desempenho ruim devido ao envio de todos fluxos desconhecidos para o controlador. Então, Devo-Flow reduz o número de casos nos quais o controlador precisa ser notificado. Nosso trabalho também reduz o número de notificações enviadas ao controlador, desde que o switch não precise notificar o controlador a cada requisição DNS que não possua fluxo instalado.

3.2 Projeto e Arquitetura do Plano de Dados

Alguns projetos são similares à nossa pesquisa por também proporem adição de funcionalidades ao plano de dados. sNICh [Ram et al., 2010] é a combinação de uma placa de interface de rede e *switching accelerator* para servidores virtualizados modernos. O software do sNICh pode suportar configuração de fluxos baseado em regras de filtros de pacotes, mas o sNICh é projetado apenas para redes virtuais, tipicamente utilizado em *Data Centers*.

P4 [Bosshart et al., 2014] propõe uma linguagem de alto nível para programar processadores de pacotes independente do protocolo. P4 também tenta aumentar o nível de abstração para o programador assim como nossa proposta. P4 é complementar a nossa proposta pois o P4 também pode ser utilizado para programar switches que possuam capacidade de realizar casamento a partir do nome de domínio. Para criar regras utilizando nomes por meio do P4, basta o operador incluir o *parser* de nomes ao fluxo de tratamento dos pacotes. Além disso, é necessário incluir as estruturas auxiliares utilizadas para mapear os endereços IPs para os nomes.

O projeto Protocol-Oblivious Forwarding (POF) [Song, 2013] possui o plano de dados altamente flexível e programável. Além disso, o POF possui o objetivo de remover qualquer dependência de um protocolo com os switches. O projeto define o termo *search key*, que é composto pela tupla <*offset, length*>. A partir da *search key* é possível definir quais campos o usuário utilizará na tabela do switch. O POF se relaciona com o nosso trabalho porque também permite realizar o *parsing* de protocolos da camada de aplicação. Mas, ele não realiza o mapeamento <IP, Nome de Domínio>. O POF é otimizado para trabalhar com protocolos que utilizam campos binários em seus cabeçalhos. Além disso, o POF pode encontrar dificuldades com protocolos que utilizam texto, por exemplo, o HTTP [Li et al., 2015] ou campos que não tem a posição pré-definida no pacote.

No artigo "Parsing Application Layer Protocol with Commodity Hardware for SDN" [Li et al., 2015], os autores defendem um *parser* de conteúdo para aumentar o poder das SDN. O objetivo é ter um controle maior sobre o tráfego. É proposto um *parser* para camada de aplicação chamado de COPY. COPY é escalável e identifica os pacotes com precisão. Este artigo se relaciona com o nosso trabalho ao também realizar o *parsing* de pacotes da camada da aplicação. Nosso diferencial em relação ao artigo é a extensão do padrão OpenFlow, e não somente a criação de uma entidade separada para ser acoplada à arquitetura SDN.

3.3 NFV

A empresa Intel, por meio do *white paper* intitulado "Service-Aware Network Architecture Based on SDN, NFV, and Network Intelligence" [Intel, 2014], propõe a extensão do padrão OpenFlow para um maior suporte para operações de DPI (Deep Packet Inspection). O DPI seria executado por um hardware NFV (*Network Functions Virtualization*) específico para a tarefa. São propostas adições de campos de meta-dados para aplicações. Mas a proposta não especifica quais dados serão extraídos. A arquitetura da Intel é intitulada de Qosmos. Nossa proposta se assemelha ao também estender o padrão OpenFlow, mas no nosso caso são especificados os campos desejados.

3.4 Middleboxes

Middleboxes são complementares à nossa solução. Como citado anteriormente, Middleboxes são dispositivos de alto custo, difícil extensão, e necessitam mão de obra especializada [Bezahaf et al., 2013]. Portanto, sua aplicação embora necessária, traz problemas para os administradores. Nossa proposta pretende simplificar o uso de Middleboxes. Propomos aplicações que possam suprir ou auxiliar a utilização de um Middlebox, realizando o encaminhamento de acordo com o nome de domínio solicitado.

Alguns exemplos de aplicações SDN substituindo Middleboxes podem ser conferidos em [Collings & Liu, 2014] e [Suh et al., 2014]. Nos dois artigos são propostos a criação de Firewall, com ou sem estado. Desta forma é provada a possibilidade de substituir um Middlebox a partir de SDN. Em nossa proposta, pretendemos criar um Firewall L7 para realizar a filtragem a partir dos nomes de domínio solicitados.

O artigo "Extending OpenFlow for Service Insertion and Payload Inspection" [Udechukwu & Dutta, 2014] possui a proposta de adicionar um dispositivo externo para fazer processamento dos pacotes (External Processing Box - EPB). O EPB possui um software *open source* para realizar o DPI. Para enviar o pacote para o EPB é criada uma nova ação e adicionada ao protocolo OpenFlow. Então, os pacotes que casam com essa ação são enviadas ao EPB para extrair as informações necessárias. Esse artigo se assemelha com o nosso projeto em também realizar DPI nos pacotes. Mas, é feito por uma entidade externa ao switch. Nosso *parser* é instalado dentro do switch.

Os artigos "Simple-fying Middlebox" [Qazi et al., 2013] e "Deep Packet Inspection as a Service" [Bremler-Barr et al., 2014] possuem propósitos similares. Os dois artigos abordam a utilização conjunta de SDN e Middleboxes. O propósito dos dois é melhorar o desempenho do tráfego através do *traffic steering*. Assim que um pacote entra na rede de um data center, este pacote é sujeito a passar por uma cadeia de Middleboxes.

Essa cadeia costuma realizar análises desnecessárias em pacotes. Portanto, através do *traffic steering* os pacotes são encaminhados apenas aos Middleboxes necessários. O fator que determina qual Middlebox o pacote deve passar são os dados do protocolo OpenFlow. A nossa proposta permitiria encaminhar os pacotes para Middlebox de acordo com o nome de domínio solicitado.

OpenBox [Bremler-Barr et al., 2015, 2016] aplica o conceito de SDN para implementar funções de rede *(Network Functions - NFs)*. Funções como Firewall, Sistema de Prevenção de Intrusos *(IPS-Intrusion Prevention System)*, Balanceador de Carga *(Load Balancer)* e Cache para Web. Essas funções são executadas por Middleboxes. Assim como esta proposta, o OpenBox também realiza DPI, análise da carga útil do pacote. No caso desta dissertação são analisados apenas os dados do pacote DNS de resposta, com o propósito de obter informações sobre nomes de domínios. O OpenBox não possui especificado o tratamento para nomes de domínio. Além disso, o OpenBox não utiliza o protocolo OpenFlow.

O artigo "Neutral Net Neutrality" [Yiakoumis et al., 2016] discute neutralidade da rede. Para isso é proposta uma forma de classificação dos fluxos. A forma de classificação de fluxos utilizada é a partir de *cookies* definidos pelos autores. Assim poderiam ser criados mecanismos e regras para aplicar nos *cookies* e criar aplicações de qualidade de serviço *(QoS)*. Os *cookies* possuem a vantagem de não precisar realizar DPI e não ferem a privacidade dos usuários. A utilização de *cookies* é vantajosa ao ser comparada por abordagens que utilizam endereço IP. *Cookies* abrangem todo o conteúdo de um Web Site. Ao criar regras para endereços IPs não é garantido que todo o conteúdo de uma página seja enquadrado nas mesmas regras. Isso se deve porque os sites podem utilizar conteúdo de vários locais para compor seu sistema. Contudo, o artigo não define como são definidos os cookies, a sua abordagem é caixa preta.

3.5 Switches de Conteúdo (L7)

Switches L7, ou Switches de Conteúdo *Content Switches* Apostolopoulos et al. [2000] ou *Switches* Kachris & Vassiliadis [2006]; Andreolini et al. [2003], são switches que interpretam cabeçalhos da camada L7 (camada de aplicação). Um exemplo de dado presente nesses cabeçalhos são nomes de domínio ou URLs. Assim é possível saber qual aplicação os pacotes se destinam. O protocolo OpenFlow até então só trabalha com cabeçalhos das camadas L2-L4 Li et al. [2015] diferentemente dos Switches de Conteúdo. Esta proposta possui semelhanças com os switches L7. Esta dissertação apresenta uma extensão para o protocolo OpenFlow interpretar dados da camada de

aplicação. Contudo, a informação interpretada é especificamente o nome de domínio.

3.6 Tabela Comparativa entre Propostas

A tabela 3.1 apresenta um comparativo entre as propostas apresentadas nesta seção. São utilizados cinco critérios de comparação. O primeiro é se a solução utiliza o paradigma SDN. O segundo parâmetro diz respeito se a solução é baseada no OpenFlow. O terceiro aspecto de comparação é se a solução disponibiliza o código aberto, ou permite implementações abertas. A quarta coluna compara a solução de acordo com a sua complexidade de implementação. A complexidade é avaliada da seguinte forma, caso sua manutenção seja difícil, ou o número de entidades envolvidas na solução são grandes ela é classificada como complexa. Por último, é levantado o custo financeiro de cada solução.

Tabela 3.1. Comparação entre os trabalhos relacionados

Projeto	SDN	OF	Código Aberto	Implementação	Custo
MPLS para SDN	Sim	Sim	N/A	Simples	Baixo
Devo-Flow	Sim	Não	Não	N/A	Não
sNICH	Sim	Sim	N/A	N/A	N/A
P4	Sim	Não	Sim	Simples	Baixo
POF	Sim	Não	Sim	Simples	Baixo
COPY	Sim	Sim	Não	Complexa	Baixo
Intel Qosmos	Sim	Sim	Sim	Simples	N/A
Middleboxes	Não	Não	Não	Complexa	Alto
OpenBox	Sim	Não	Sim*	Complexa	N/A
SDN Stateful Firewall	Sim	Sim	Não	Complexa	Baixo
EPB	Sim	Sim	Sim	Simples	Baixo
Switches de Conteúdo	Não	Não	Não	Complexa	Alto
Neutral Net Neutrality	Não	Não	Não	Complexa	N/A
DN+OpenFlow	Sim	Sim	Sim	Simples	Baixo

A solução proposta nesta dissertação, DN+OpenFlow, utiliza o paradigma SDN e o padrão OpenFlow. Possui código aberto disponível no Github e implementação simples. O seu custo é considerado baixo pois não é necessária nenhuma alteração adicional, controlador e switch já estão aptos para aceitar o novo padrão desenvolvido. Já a solução mais antagônica nesta tabela são os Middleboxes. Middleboxes são estruturas caras, tanto na implementação quanto na manutenção, e possuem arquitetura proprietária e defasada.

Capítulo 4

DN+OpenFlow

Este capítulo apresenta o DN+OpenFlow. O DN+OpenFlow é uma nova versão do OpenFlow, capaz de criar regras de fluxo utilizando nomes de domínio. Para possibilitar essa solução é necessário estender o protocolo OpenFlow já existente. Esta extensão consiste em adicionar dois campos à estrutura de casamento do protocolo OpenFlow: os nomes de domínio de origem e destino.

A arquitetura proposta nessa dissertação aproveita os dados que já circulam pela rede para aumentar o poder de abstração do switch. Para isso, é criada uma estrutura auxiliar para armazenar os endereços IPs e os nomes de domínio relacionados.

A seção 4.1 apresenta como é modelado o projeto, e a sua arquitetura, ou seja, o que é o DN+OpenFlow. A seção 4.2 dá detalhes técnicos de como o projeto foi implementado. Nesta seção é detalhada a implementação do switch DN+OpenFlow. A seção 4.3 detalha como foi a implementação do projeto no controlador POX. A seção 4.4 apresenta as aplicações possíveis utilizando o DN+OpenFlow.

4.1 Arquitetura DN+OpenFlow

Nesta seção serão apresentados os pontos que compõem a arquitetura do DN+OpenFlow. O sistema DN+OpenFlow abrange a arquitetura do switch e do controlador. Serão detalhados nessa seção os passos realizados para alcançar os objetivos desejados.

A figura 4.1, representa para um padrão de acesso em específico, como a arquitetura do DN+OpenFlow funciona. Considerando uma operação proativa do switch, primeiro, o usuário do controlador insere as regras **(1)** utilizando nomes de domínio para um web site específico. Quando o cliente realiza uma solicitação DNS **(2)** e o servidor DNS **(3)** responde com uma resposta, o switch OpenFlow interpreta a res-

posta e preenche a tabela com os respectivos endereços IPs e nomes de Domínio **(4)**. Os próximos pacotes IPs para a conexão entre cliente e servidor irão carregar o endereço IP para aquele servidor. Estes pacotes serão consultados na tabela <IP, Nome de Domínio>, assim os nomes de domínio são associados para cada pacote, podendo ser comparados na tabela de fluxos do switch. Um casamento irá ocorrer para a regra inserida anteriormente definida pelo controlador, e o fluxo será tratado de acordo com a regra inserida. Por último, o usuário realiza o acesso ao servidor desejado **(5)**.

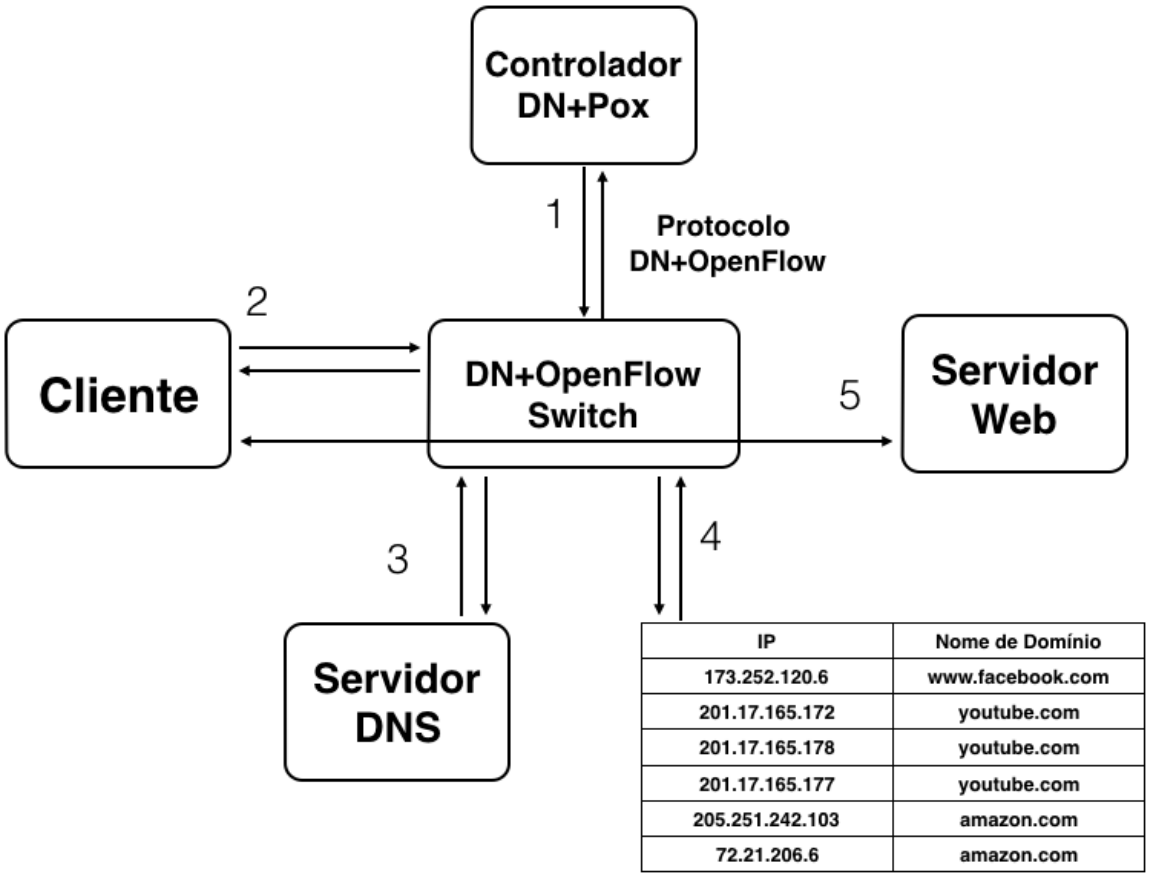


Figura 4.1. Exemplo de como a arquitetura do DN+OpenFlow funciona

Para possibilitar que esta proposta trabalhe com nomes de domínio associado a fluxos, a arquitetura deste projeto consiste nos seguintes componentes apresentados nas sub-seções seguintes.

4.1.1 Alteração na Tabela de Fluxos

Uma entrada de fluxo no protocolo OpenFlow é composto por três estruturas. A primeira estrutura representa os cabeçalhos dos campos utilizados para realizar o casamento. A segunda estrutura representa os contadores estatísticos associados àquele

fluxo. São contabilizados por exemplo número de pacotes e a quantidade bytes que aquela entrada de fluxo possui. A última estrutura representa as ações associadas ao fluxo. As três ações obrigatórias são, porta de saída para encaminhamento do pacote, descarte e o envio para o controlador. A estrutura de fluxo foi detalhada na figura 2.2.1.

Para viabilizar este projeto é necessário realizar a alteração na estrutura de cabeçalhos da arquitetura OpenFlow. A estrutura de casamento composta pelos cabeçalhos dos pacotes é um sub-estrutura da estrutura de fluxo. A versão 1.0 do protocolo OpenFlow possui 12 campos de cabeçalho para casamento. Estes campos correspondem aos dados dos cabeçalhos dos pacotes das camadas L2-L4.

Esta solução modificou a estrutura de cabeçalhos para incluir os campos nome de domínio de origem e destino. Neste caso, os campos são virtuais, pois não existem nos cabeçalhos dos pacotes. Assim, é possível criar regras de casamento que serão comparadas com os fluxos. A figura 4.2 apresenta a adição dos novos campos à estrutura de cabeçalhos. (Apesar desta proposta abordar o protocolo OpenFlow 1.0, a mesma abordagem pode ser aplicada para outras versões do protocolo).

Figura 4.2. Cabeçalhos de casamento após a inserção dos campos DN_SRC e DN_DST (Nome de Domínio)

4.1.2 Inspeção do pacote DNS de resposta

Este projeto aproveita os pacotes DNS de resposta para armazenar os endereços IPs e os nomes de domínio aos quais eles pertencem. Assim, não é necessário realizar uma consulta para descobrir essa relação. São utilizados os dados que trafegam pelo switch para realizar o mapeamento <Endereço IP, Nome de Domínio>.

Esta tarefa consiste na interpretação *(parser)* do pacote DNS de resposta para extrair os endereços IPs e nomes de domínio relacionados. Sempre que um pacote DNS de resposta passa pelo switch, o switch é responsável por inspecionar o pacote. Apenas as respostas do tipo A são tratadas pelo DN+OpenFlow. Respostas do tipo são responsáveis por conter os endereços IPv4. Outro tipo de respostas não são tratados, como por exemplo as do tipo CNAME. Respostas do tipo CNAME não são tratadas pelo DN+OpenFlow.

São coletados dos pacotes DNS de resposta os endereços IP, os nomes de domínio associados e o TTL *(time to live)* do pacote DNS. Então, são armazenadas as tuplas <Endereço IP, Nome de Domínio, TTL> em uma tabela para futuras consultas.

4.1.3 Tabela <Endereço IP, Nome de Domínio>

A tabela 4.1 ilustra a relação <Endereço IP, Nome de Domínio> que é armazenada pelo switch. Esta tabela precisa armazenar todos nomes de domínio acessados em uma rede empresarial.

Tabela 4.1. Tabela Endereço IP, Nome de Domínio.

Endereço IP	Nome de Domínio	TTL
173.252.120.6	`www.facebook.com`	255
201.17.165.172	`www.youtube.com`	255
201.17.165.178	`www.youtube.com`	255
201.17.165.177	`www.youtube.com`	255
201.17.165.157	`www.youtube.com`	255
176.32.98.166	`www.amazon.com`	255
205.251.242.103	`www.amazon.com`	240
72.21.206.6	`www.amazon.com`	240

A tabela <IP, Nome de Domínio> possui o campo TTL *(**T**ime **T**o **L**ive)* para evitar que a informação na tabela se torne defasada, podendo causar erro no roteamento. Isto pode ocorrer em casos em que o endereço IP sofra qualquer tipo de alteração. O campo TTL funciona da seguinte forma, a cada segundo é decrescido seu valor. Quando esse valor do campo se iguala a zero o registro é retirado da tabela. O campo TTL é uma forma de manter a tabela atualizada. Outra funcionalidade do campo TTL é evitar o gasto excessivo de memória.

4.1.4 Mapeamento pacote para nome de domínio

O mapeamento pacote para nome de domínio é uma função importante para a arquitetura. Sem ela, seria impossível identificar o nome de domínio de cada pacote. Para realizar a associação aos nomes de domínio é necessário consultar a tabela <IP,Nome de Domínio>. Assim, é realizada uma pesquisa pelo endereço IP do pacote, caso exista algum nome de domínio associado, este nome é atribuído ao fluxo.

4.1.5 Função de casamento da tabela de fluxos

É necessário modificar a função de casamento de fluxos do switch OpenFlow. Os dois novos campos da estrutura de casamento precisam ser comparados entre os fluxos, no caso os campos nomes de domínio de origem e destino (`dn_src/dn_dst`). Estes dois campos precisam ser comparados com os nomes de domínio de origem e destino atribuídos a partir do endereço IP dos pacotes. Sempre que um casamento ocorrer no campo nome de domínio, o switch é capaz de executar as ações associadas àquele fluxo. São aceitos apenas casamentos exatos, *wildcards* não são suportados.

4.1.6 Inserção de regras de casamento utilizando Nomes de Domínio

É necessário prover uma forma para o controlador inserir regras de casamento utilizando nomes de domínio na tabela de fluxos do switch OpenFlow. Primeiramente, o controlador deve ser capaz de lidar com regras utilizando nomes de domínio. É necessário também estender o protocolo OpenFlow utilizado para comunicar com o controlador e o switch. Então, poder carregar regras utilizando nomes de domínio do controlador para o switch.

4.2 Implementação

Para implementação do DN+OpenFlow foi escolhida a implementação de Stanford para o protocolo OpenFlow escrita na linguagem de programação C [1]. O foco desta implementação é a pesquisa. Esta implementação foi utilizada pela facilidade de depuração e alteração de código. Foi utilizada a versão 1.0 do protocolo OpenFlow. Essa distribuição é utilizada pelo emulador SDN Mininet [2] como switch padrão funcionando em espaço de usuário. O código fonte também está disponível na distribuição do Mininet [3]. O código do DN+OpenFlow está disponível no Github [4]. As principais alterações realizadas também estão no apêndice da dissertação.

Foram modificados 7 arquivos e um arquivo foi criado. A figura 4.3 apresenta a estrutura de arquivos do OpenFlow 1.0 Stanford Reference e os arquivos modificados e criados. As modificações realizadas estão descritas nas subseções seguintes.

[1] `http://archive.openflow.org/wp/downloads/`
[2] https://mininet.org
[3] https://github.com/mininet/openflow
[4] https://github.com/lucasmaiasilva/DN-OpenFlow

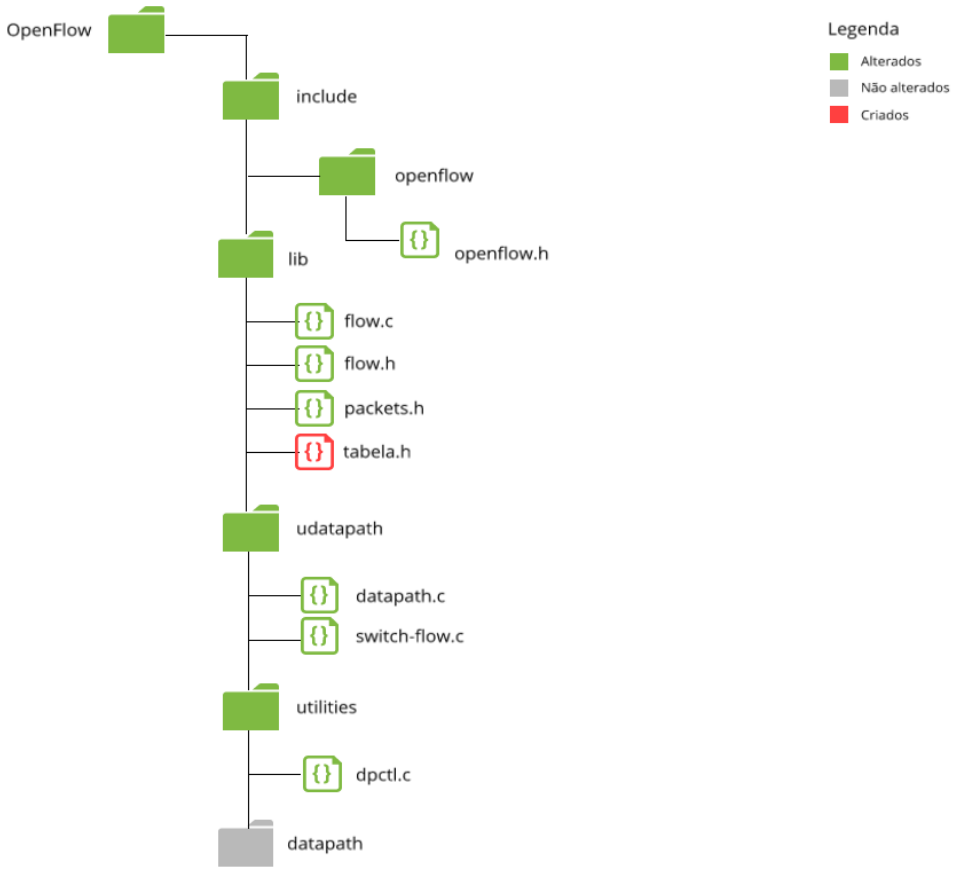


Figura 4.3. Estrutura de diretórios do switch OpenFlow Stanford Reference

4.2.1 Estruturas de fluxo e casamento do OpenFlow

As várias estruturas que operam os fluxos do protocolo OpenFlow são definidas no arquivo `openflow.h`. Estas estruturas contêm os campos de cabeçalho de casamento *(header fields)*. Os cabeçalhos para casamento são representados pela estrutura `ofp_match` no código do switch. Esta estrutura contém a tupla com 12 campos dos cabeçalhos das camadas L2-L4. A estrutura `ofp_match` possui 40 bytes no total.

Foram adicionados à estrutura `ofp_match` os campos para armazenar os nomes de domínio de origem e destino. Os campos adicionados são `dn_src` (nome de domínio de origem) e `dn_dst` (nome de domínio de destino) com 256 bytes cada. O valor de 256 bytes foi escolhido pois é o tamanho máximo permitido para nomes de domínio. No total, a nova estrutura passou a possuir 552 bytes e 14 campos para casamento. É importante ressaltar que ao final de cada estrutura existe uma função para checar o seu tamanho em bytes. Para isso é utilizado o método `OFP_ASSERT`.

Todas as estruturas de fluxo que importam a `ofp_match` precisam ser corrigidas para o novo tamanho da estrutura. As estruturas alteradas foram:

`ofp_flow_mod`, `ofp_flow_removed`, `ofp_flow_stats_request`, `ofp_flow_stats` e `ofp_aggregate_stats_request`.

4.2.2 Inspeção de pacotes DNS de resposta

O DN+OpenFlow utiliza os dados dos pacotes DNS de resposta para formar a relação <IP, Nome de Domínio>. O DNS de resposta é escolhido devido aos dados existentes no *payload*, no caso os endereços IPs. Para isso, foi criado um interpretador *(parser)* de pacotes DNS de resposta. O protocolo OpenFlow já possui o interpretador dos 12 campos existentes na estrutura de casamento (`ofp_match`).

O novo interpretador foi adicionado quando são identificados pacotes do protocolo UDP utilizando a porta 53. De acordo com a RFC1035 esses pacotes pertencem ao serviço de DNS [Mockapetris, 1987]. O interpretador do switch OpenFlow fica no arquivo `flow.c`. No arquivo `flow.h` são definidas as estruturas necessárias para realizar a interpretação dos cabeçalhos dos pacotes. A função principal do interpretador do switch é a `flow_extract()`. Nela ocorre todo o *pipeline* de extração de cabeçalhos dos pacotes no switch.

Os cabeçalhos dos pacotes das camadas L2-L4 foram definidos no arquivo `packets.h`. As estruturas dos cabeçalhos são utilizadas para extrair as informações corretas do *buffer* que armazena os pacotes. O processo funciona de maneira simples, a partir do protocolo identificado é possível saber a quantidade exata de bytes necessária para obter a informação e assim popular a estrutura de fluxo.

Foram adicionadas ao arquivo `packets.h` três estruturas para armazenar os dados dos pacotes DNS de resposta. São elas `dns_header`, `dns_question` e `dns_ans_header`. A primeira estrutura armazena os dados do cabeçalho da resposta DNS. A segunda estrutura armazena os dados da consulta DNS realizada, o nome de domínio é armazenado a parte. Esta decisão foi devido ao seu tamanho variável. Por último, a estrutura `dns_ans_header` armazena os dados da resposta DNS obtida.

A figura 4.4 aborda como funciona o interpretador de pacotes DNS de resposta. O número **1** representa a identificação do pacote DNS de reposta (Porta UDP de origem 53). O número **2** realiza a extração dos dados do cabeçalho do pacote. São extraídos os campos de identificação *id*, *flags*, quantidade de consultas DNS, quantidade de respostas, quantidade de respostas autoritativas, quantidade de registros adicionais no pacote. No número **3** é realizada a extração dos dados da consulta, o nome de domínio consultado é armazenado para ser adicionado à tabela junto com o endereço IP e o TTL do pacote. Além do nome de domínio são extraídos os tipos e a classe da consulta DNS. O número **4** realiza a extração dos endereços IPv4 das respostas. Esta

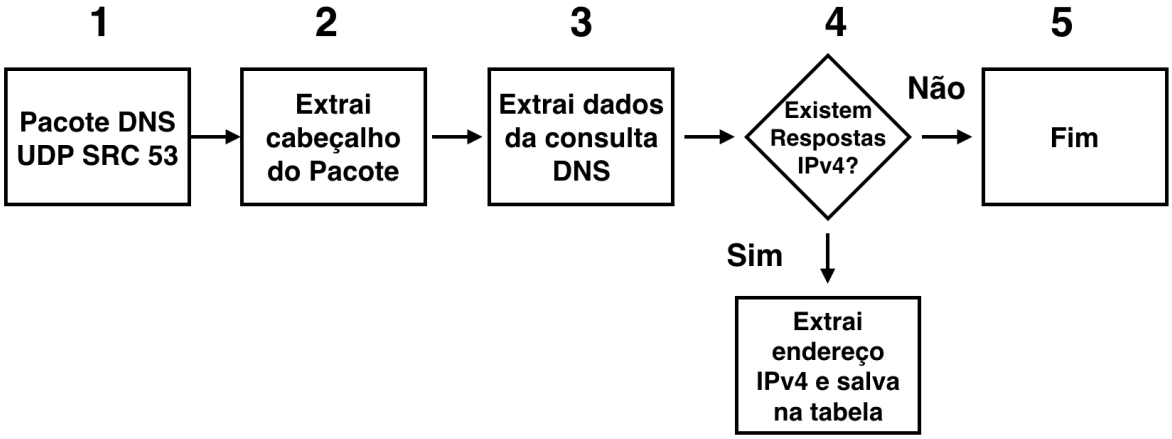


Figura 4.4. Interpretador dos pacotes DNS de resposta

etapa consiste na análise do *payload* do pacote DNS. Além do endereço IPv4 é extraído também o TTL do pacote. Esta operação se repete em todos os registros do pacote que sejam do tipo endereço IPv4. Após terminar esta etapa o interpretador insere na tabela <IP,Nome de Domínio> a tupla referente àquele pacote junto de seu TTL. Por último, o interpretador chega à etapa **5** onde termina sua execução.

4.2.3 Tabela IP, Nome de Domínio

Para armazenar a relação <IP, Nome de Domínio> foi criada uma tabela baseada em 4.1. A estrutura da tabela está definida no arquivo `tabela.h`. Além da tabela, foram criadas as funções para realizar as operações na tabela. A figura 4.5 apresenta dois diagramas para o arquivo `tabela.h`. O diagrama da esquerda ilustra os campos presentes na tabela. O diagrama da direita exibe as funções utilizadas para operá-la.

O campo `uint8_t DN[256]` é utilizado para armazenar o nome de domínio. O campo `uint32_t IP` é utilizado para armazenar os endereços IPv4. O campo `uint32_t TTL` armazena o valor do TTL atual para o registro. O campo `uint32_t OLD_TTL` é responsável por armazenar o TTL inicial do pacote DNS.

A função `void adiciona()` adiciona registros na tabela. A função `void apaga()` remove os registros da tabela. A função `int tabelaCmpIp()` realiza a comparação de registros pelo campo IP. A função `uint8_t* buscaIp()` busca na tabela por endereço IP e retorna o registro encontrado. A função `int busca()` apenas checa se um registro existe na tabela, retornando 1 caso encontrado e 0 caso contrário. A função `void imprimeTabela()` possui o objetivo apenas para depuração e imprime no arquivo o conteúdo da tabela <IP, Nome de Domínio>.

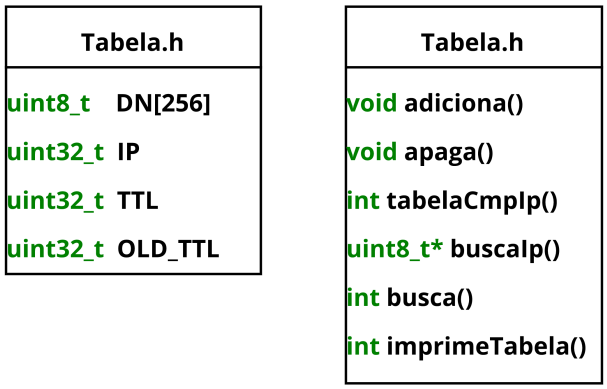


Figura 4.5. Campos e funções da estrutura tabela <IP,Nome de Domínio>

A tabela é declarada estaticamente no arquivo `flow.c`. Assim, ela possui valor global durante a execução do switch. Ela é mantida ordenada pelo campo IP, utilizando a função `qsort()` padrão da linguagem C. Para realizar consultas na tabela é utilizada a busca binária padrão da linguagem C `(bsearch())`. O que evita buscas sequenciais, diminuindo o custo adicional *(overhead)* na aplicação. Cada registro na tabela ocupa 352 bytes na memória. O tamanho máximo da tabela é escolhido pelo usuário de acordo com a sua necessidade.

Cada registro possui o campo `TTL` (quantidade de segundos para a informação contida no pacote expirar). Esse campo inicialmente possui valor inicial do TTL do pacote DNS o qual o nome de domínio e endereço IP foram extraídos. É implementado no código do switch uma função para decrescer os segundos do registro. Para isso, foi utilizada a função `SIGALARM` de C, gerando um alarme para decrescer 1 segundo de cada registro na tabela. Assim que um registro atinge o valor 0 no TTL, ele é eliminado da tabela.

4.2.4 Mapeamento Pacote para Nome de Domínio

O próximo passo é associar nomes de domínio aos pacotes que entram no switch. Para isso é necessário saber qual o seu endereço IP de origem ou destino. Por consequência, pacotes que possuem apenas dados da camada L2 não possuem nome de domínio associado. No arquivo `flow.c` dentro da função `flow_extract` é realizado o parser do pacote. Existe uma etapa do parser que são retirados os cabeçalhos do protocolo IP. Neste momento o pacote atribui seu endereço IP de origem e destino ao fluxo.

Após o fluxo já possuir endereços IP de origem e destino, estes endereços são

utilizados para consultar a tabela <IP, Nome de Domínio>. Caso seja encontrado algum nome de domínio para o endereço IP buscado, o mesmo é atribuído ao fluxo. Quando ocorre um *hit* na tabela <IP,Nome de Domínio> o campo `TTL` é atualizado para o valor do campo `OLD_TTL`. Assim, é possível evitar que o registro na tabela expire e o fluxo fique sem nome de domínio atribuído.

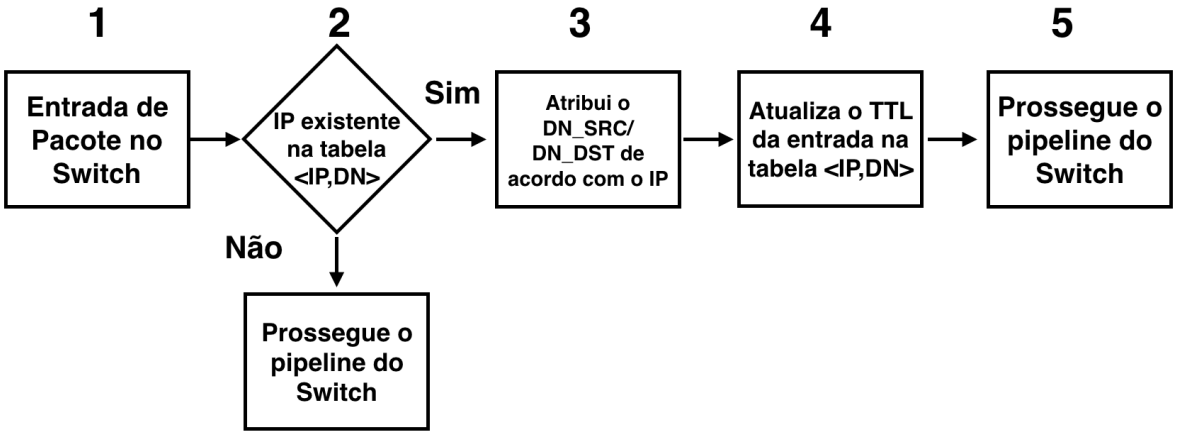


Figura 4.6. Mapeamento de pacotes IP

A figura 4.6 demonstra como funciona o mapeamento dos pacotes. Em **1** é representada a entrada de pacotes que possuem endereços IPs associados. Em **2** é consultado se alguns dos seus endereços IPs (origem/destino) possuem entrada na tabela <IP, Nome de Domínio>. Caso negativo, o switch prossegue com o pipeline comum. Em caso positivo(**3**), são atribuídos os nomes de domínio de origem ou de destino de acordo com o endereço IP. O número **4** representa a atualização do campo TTL para a entrada na tabela. Por último, em **5** o switch prossegue com seu *pipeline* comum. O DN+OpenFlow atribui nomes de domínio a qualquer protocolo que funcione sobre o protocolo IP. A única restrição é que já tenha ocorrido alguma requisição DNS para a relação IP, Nome de Domínio. Assim, qualquer pacote que utilize IP pode possuir um nome de domínio associado.

4.2.5 Inserção de regras utilizando Nomes de Domínio

O protocolo foi estendido para lidar com os novos campos utilizados para a identificações de fluxos, e o switch foi alterado para entender as novas regras e os novos campos para então processá-los e lidar com a tabela de fluxos estendida.

Para permitir a criação de regras utilizando nomes de domínio, existem duas formas. Uma é estendendo o protocolo DN+OpenFlow para a ferramenta DPCTL.

Outra solução possível é também estender o DN+OpenFlow para o código de algum controlador OpenFlow. Ambas soluções foram abordadas. A extensão do protocolo OpenFlow para o POX esta descrita na seção 4.3.

O DPCTL é uma ferramenta de depuração, a qual permite a inserção de fluxos individuais em um switch OpenFlow. Além de inserção de fluxos é possível requisitar dados estatísticos sobre fluxos, apagar a tabela de fluxos do switch e até mesmo executar utilitários de *benchmark* que existem no DPCTL. O código fonte do DPCTL é escrito em C e sua implementação vem junto com a implementação do switch OpenFlow. Seu código fonte pode ser encontrado no arquivo `dpctl.c`.

A ferramenta DPCTL foi modificada para permitir a inserção de regras utilizando nomes de domínio. Seu funcionamento é simples, a função `str_to_flow()` é responsável por capturar os dados preenchidos pelo usuário e transformá-los em uma entrada de fluxo e enviá-los ao switch. Para isso, foram adicionados dois campos de tratamento nesta função. Os campos `dn_src/dn_dst` foram adicionados. Assim, é possível inserir entradas de fluxo utilizando nomes de domínio. Um exemplo de como utilizar a nova funcionalidade pode ser visto abaixo:

```
dpctl add-flow tcp:localhost:6634
dn_dst=www.facebook.com,priority=65535,idle_timeout=65535,actions=
```

O comando `add-flow` é utilizado para adicionar uma entrada de fluxo à tabela do switch. A *string* `tcp:localhost:6634` serve para identificar qual switch conectar. O protocolo TCP é utilizado para troca de dados entre o DPCTL e o switch. O endereço *localhost* é onde o switch está sendo executado. Neste caso no mesmo endereço IP do programa DPCTL. A porta `6634` é a porta reservada para comunicação entre o switch e o programa DPCTL. Vale lembrar que a porta `6633` é a porta padrão de conexão entre o switch e o controlador. A *string* `dn_dst=www.facebook.com` define qual nome de domínio de destino o fluxo se aplica. A string `priority=65535` define o nível de prioridade do fluxo, neste caso o máximo. A string `idle_timeout` determina o período máximo de inatividade em segundos para o fluxo ser removido da tabela, no caso 65535 segundos. Por último, é tratada qual ação é associada ao fluxo. Neste caso, quando o campo `actions=` está em branco nenhuma ação é associada. Logo, os pacotes deste fluxo serão descartados *(Drop Action)*.

4.2.6 Função de Casamento

Para possibilitar o casamento de regras utilizando nomes de domínio, foi adicionada à função de casamento de fluxos a comparação entre nomes de domínio. A função

de casamento é uma função de comparação entre os fluxos que passam pelo switch e por uma dada regra presente na tabela. O nome da função responsável pelo casamento é `flow_fields_match()`, e esta função está no arquivo `switch-flow.c`. A função `flow_fields_match()` retorna 1 quando um valor é encontrado e 0 em caso contrário.

4.3 DN+POX

Para completar a implementação do DN+OpenFlow, foi necessário adaptar um controlador para a nova versão do OpenFlow criada. O controlador utilizado foi o POX. O POX é um controlador SDN para a versão 1.0 do protocolo OpenFlow. O POX é voltado para pesquisas devido à sua facilidade de uso. A linguagem de programação utilizada pelo POX é a Python. Python é uma linguagem de alto nível e com grande poder de abstração. Os programas escritos em Python são interpretados. Logo, seu desempenho é inferior quando comparados com controladores utilizados em produção, que utilizam linguagens compiladas. Contudo, a escrita dos programas para o controlador POX é mais simples e prática. A sua escolha para este projeto foi influenciada devido a suas características, e grande adesão no meio acadêmico.

A nova versão gerada do POX foi batizada de DN+POX. O seu código fonte está disponível no seguinte repositório no GitHub [5]. A versão do POX utilizada para a criação do DN+POX foi a 0.20 intitulada de Carp, encontrada no seguinte link do GitHub [6]. A versão do Python utilizada é a 2.7. O POX pode ser utilizado nas plataformas Windows, Linux e Mac OS. O protocolo OpenFlow utilizado como base é o 1.0. Tanto o DN+POX e DN+OpenFlow foram escritos baseados no OpenFlow versão 1.0.

4.3.1 Estrutura de Diretórios do POX

O código do controlador possui dois arquivos principais que regem o comportamento do protocolo OpenFlow. Esses dois arquivos são o `libopenflow_01.py` responsável por definir as mensagens trocadas entre controlador e switch, e o `of_01.py` responsável por definir os eventos OpenFlow que o POX atende. Como serão alteradas as mensagens utilizadas no protocolo OpenFlow, todas as alterações realizadas neste trabalho foram no arquivo `libopenflow_01.py`. A figura 4.7 representa o caminho seguido para realizar a alteração no arquivo `libopenflow_01.py`.

[5] https://github.com/lucasmaiasilva/DN-pox
[6] https://github.com/noxrepo/pox

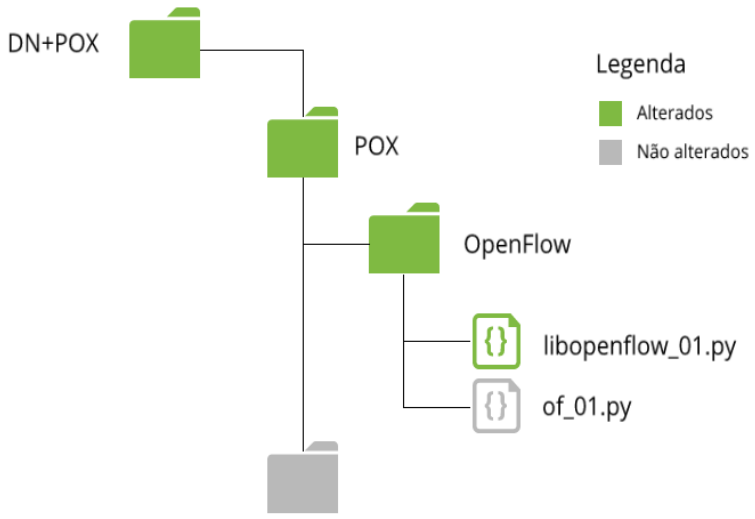


Figura 4.7. Árvore de diretórios do POX

As mensagens e outras estruturas utilizadas no protocolo OpenFlow estão definidas no arquivo `libopenflow_01.py`. Cada mensagem possui sua respectiva classe definida nesse arquivo. A única classe alterada nesse projeto foi a `ofp_match`. A classe `ofp_match` é responsável por tratar os campos de cabeçalho dos pacotes utilizados para casamento. Outra classe importante é `ofp_flow_mod`, esta classe é responsável pela inserção de regras na tabela de fluxos do switch. A `ofp_flow_mod` não sofreu alterações, mas ela possui todos os dados da `ofp_match`. A estrutura de casamento é uma das estruturas que compõem uma entrada de fluxo, como ilustrado na figura 2.2.1.

4.3.2 As classes `OFP_MATCH` e `OFP_FLOW_MOD`

A classe `ofp_match` utiliza o dicionário `ofp_match_data` que é responsável por mapear os 12 campos de cabeçalho para casamento do protocolo OpenFlow versão 1.0. Foram adicionados à essa estrutura os dois novos campos utilizados, `dn_src` e `dn_dst`, totalizando 14 campos. O tamanho total de um objeto `ofp_match` passa de 40 bytes para 552 bytes.

Os métodos alterados nessa classe são: `from_packet()`, `__len__()`, `pack()` e `unpack()`. O método `from_packet()` é utilizado para realizar a interpretação dos cabeçalhos dos pacotes que chegam ao controlador. Em termos técnicos do switch OpenFlow, esta função é responsável por preencher a estrutura `ofp_match` a partir de um dado pacote. Esta ação é necessária quando um pacote não é identificado no switch ele é enviado ao controlador. O evento gerado nesse caso é o `packet_in`. Em eventos deste tipo não estão disponíveis as informações de casamento do fluxo obtidas

no switch. Logo, o *parser* realizado no switch é refeito no controlador. A versão 1.2 do protocolo OpenFlow já possui a estrutura de casamento dentro das mensagens do tipo `packet_in`, o que evita repetir a interpretação dos cabeçalhos dos pacotes. Portanto, o mesmo interpretador implementado no switch deve ser portado para o controlador. Contudo, a sua implementação é bem mais simples. O Pox já provê um parser para pacotes DNS, e a tabela <IP, Nome de domínio> foi implementada utilizando um dicionário Python. Ao final da execução da função `from_packet()` a tabela <IP, Nome de Domínio> é consultada em busca de nomes de domínio associados aos endereços IPs do pacote, em caso positivo é atribuído o nome de domínio relativo àquele IP. Assim, os dados dos cabeçalhos e os nomes de domínio já estão associados aos campos de casamento da classe `ofp_match`. Desta forma o controlador pode instalar a nova regra de fluxo no switch.

O método `__len__()` é responsável por retornar o tamanho do pacote (número fixo de bytes). O tamanho do pacote influencia diretamente no funcionamento do POX. Pacotes com tamanho errado não são processados pelo controlador, levantando uma exceção e tratamento de erro. É importante ressaltar que o tamanho dos pacotes (bytes) trafegados influi no funcionamento correto do controlador.

A função `pack()` possui como objetivo transformar o objeto da classe em uma *string* de bits para ser enviada ao switch. A função `unpack()` faz o contrário, transformando *string* de bits no objeto desejado. No caso da função `pack`, caso o casamento não utilize nenhum nome de domínio é necessário realizar o *padding* do pacote. O *padding* acrescenta 64 bytes (*string* binária de 0s) ou até 1 byte ao pacote. Esta ação depende de quantos campos de nome de domínio estão preenchidos. Outro fator é o tamanho do nome de domínio. Por exemplo, um nome de domínio que possua apenas 10 bytes, precisa que o *padding* complete com 22 bytes para fechar o tamanho correto do campo. A figura 4.8 apresenta os principais métodos que participam da modificação do DN+POX.

A classe `ofp_flow_mod` é responsável pelas mensagens de inserção de fluxos na tabela do switch. A classe `ofp_flow_mod` contém a classe `ofp_match`. Assim o novo fluxo que será instalado deve respeitar os dados contidos na classe `ofp_match`. Além dos campos utilizados pelo casamento, a classe `ofp_flow_mod` possui os metadados do fluxo. Alguns deles são a prioridade do fluxo, os dois *timeouts (hard timeout e idle timeout)*, e outros campos descritos na estrutura de fluxos.

Desta forma todos os programas escritos para o Pox já podem ser utilizados e já estão compatíveis com os novos tamanhos das mensagens alteradas. Os módulos do Pox de switch de aprendizado, `l2_learning`, `l3_learning`, já disponibilizados pelo código do POX funcionam corretamente com a nova biblioteca modificada. Vale destacar

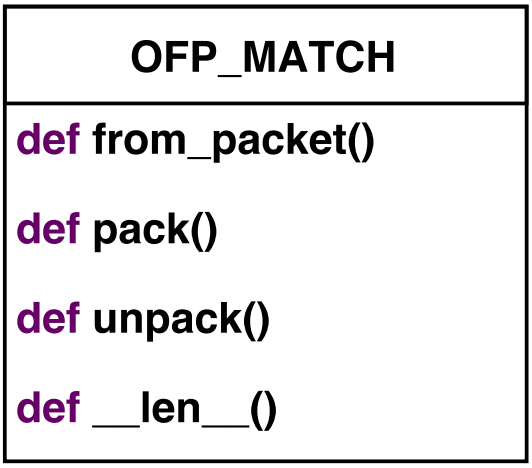


Figura 4.8. Métodos alterados da classe `OFP_MATCH`

que o arquivo alterado (`libopenflow_01.py`) é utilizado por todos os programas que utilizam o protocolo OpenFlow. Um exemplo de utilização da nova versão do DN+POX pode ser conferido no apêndice B e no repositório do projeto [7].

De uma forma genérica um roteiro de alterações no POX para estender o protocolo OpenFlow pode ser enumerado da seguinte forma:

- Adicionar os campos ao dicionário `OFP_MATCH_DATA`.
- Adicionar o parser dos pacotes DNS de resposta e a tabela <IP, Nome de Domínio> à função `from_packet()`.
- Consultar a tabela <IP, Nome de Domínio> e atribuir os nomes de domínio de acordo com os endereços IPs.
- Alterar o método `lenght` para as classe `OFP_MATCH`.
- Alterar o método `pack` e `unpack` da classe `OFP_MATCH` realizar o *padding* caso necessário.

4.4 Aplicações

Nesta seção, serão descritas algumas aplicações que podem ser utilizadas através da extensão do protocolo OpenFlow proposta. Essas aplicações utilizam regras de encaminhamento utilizando nomes de domínio e as ações pré-definidas do padrão OpenFlow. As ações permitidas são: descarte, encaminhamento (porta de saída), e o envio de

[7] github.com/lucasmaiasilva/dn-pox

pacotes para o controlador. A figura 4.9 demonstra como implementar regras de fluxo com ou sem o DN+OpenFlow.

#	In Port	IP SRC	IP DST	L4 SRC	L4 DST	DN SRC	DN DST	Ações
1	*	*	x.x.x.5	*	*	*	*	Descarte
2	*	*	x.x.x.6	*	*	*	*	Descarte
3	*	*	*	*	*	*	malicious.com	Descarte
4	*	*	*	*	*	*	www.facebook.com	Porta 2
5	*	*	*	*	*	*	youtube.com	Porta 2
6	*	*	*	*	*	*	ieeexplore.ieee.org	Porta 1
7	*	*	201.17.165.172	*	*	*	*	Descarte
8	*	*	201.17.165.177	*	*	*	*	Descarte
9	*	*	201.17.165.178	*	*	*	*	Descarte
10	*	*	*	*	*	*	youtube.com	Descarte
11	*	*	*	*	*	*	audio.foo.org	Porta 1
12	*	*	*	*	*	*	image.foo.org	Porta 2
13	*	*	*	*	*	*	video.foo.org	Porta 3
14	*	*	*	*	*	baixaki.com.br	*	Porta 4

Figura 4.9. Tabela de fluxos do switch

4.4.1 Regras de casamento persistentes

Regras de casamento que utilizam nomes de domínio podem lidar com tráfego para um dado servidor caso ocorra troca de endereço IP. Por exemplo, um *malware* se conecta ao servidor para realizar transferência de dados do usuário utilizando um nome de domínio associado a um endereço IP dinâmico. O administrador de rede pode descobrir o endereço IP atual do *malware* e inserir uma regra para bloquear seus ataques, como demonstrado na figura 4.9 pelas linhas **1** e **2**.

Esta abordagem parece sólida, mas o hacker pode alterar os endereços IP dos seus servidores para evitar que o seu *malware* seja bloqueado. Basta ele mudar os endereços IPs associados aos nomes que o *Malware* utiliza. Portanto, o administrador de redes pode simplesmente adicionar uma regra para bloquear o nome de domínio utilizando o DN+OpenFlow e ignorar as alterações de endereço IP, como demonstrado na figura 4.9 pela linha **3**.

4.4.2 QoS *(Quality of Service)*

Uma aplicação QoS utilizando políticas baseadas em nomes de domínio pode ser criada utilizando o DN+OpenFlow. Em um ambiente de rede hipotético onde existem dois links de Internet, um utilizado para o tráfego de entretenimento, como Facebook ou Youtube, e outro link utilizado para o tráfego utilizado para produção. Esta separação é utilizada para não perturbar o desempenho de aplicações críticas. Então, na porta 1 do switch, existe um link dedicado com baixa largura de banda para acessar Youtube, Facebook, e outro link com uma melhor vazão é disponibilizado na porta 2, dedicado para o tráfego de aplicações críticas. As linhas **4, 5 e 6** da figura 4.9 demonstram como as regras inseridas seriam.

4.4.3 Firewall

A criação de um Firewall para Web sites é mais simples utilizando nomes de domínio do que utilizando apenas endereços IPs. DN+OpenFlow evita que seja necessária a inserção de todos os endereços IPs para um site em específico. Por exemplo, as linhas **7, 8 e 9** da figura 4.9 demonstram o exemplo de regras considerando apenas três endereços IPs para o Youtube. (Repare que o Youtube possui muito mais que três endereços IPs). O Firewall também pode se comportar como um AdBlock. Domínios conhecidos que realizam propagandas podem ser bloqueados. Alguns deles como doubleclick.net e domínios do *Google Adservice*.

A linha **10** da figura 4.9 demonstra como seria para bloquear todos endereços IPs do Youtube utilizando o DN+OpenFlow. Fica claro a quantidade de regras que são economizadas na tabela, tornando a tarefa de bloquear Web Sites muito mais simples. Esta abordagem também salva espaço na memória do switch, desde que são utilizadas uma quantidade menor de regras.

4.4.4 Gerenciamento de Web Sites

Um exemplo de uma aplicação para gerenciamento de web sites é o redirecionamento dos pacotes dentro de um data center para os servidores corretos. O switch OpenFlow pode encaminhar cada pacote para o servidor correto de acordo com o nome de domínio requisitado. As linhas **11, 12 e 13** da figura 4.9 demonstram as regras para um *data center* com três tipos de servidores de aplicações: áudio, vídeo e imagens. Cada servidor está conectado a uma porta do switch. Os pacotes são encaminhados de acordo com o nome de domínio associado aos pacotes.

4.4.5 *Traffic Steering*

O DN+OpenFlow pode ser utilizado junto com Middleboxes. A técnica de *traffic steering* consiste em utilizar SDN e Middleboxes em conjunto. Os fluxos podem ser caracterizados de acordo com o nome de domínio requisitado e assim encaminhado para a aplicação necessária. Desta forma, garantindo que apenas os pacotes que precisam ser inspecionados por aplicações específicas sejam processados. Assim é melhorado o desempenho da rede. As aplicações possíveis de serem utilizadas são anti-vírus, Sistemas de prevenção/detecção de intrusos (IPS/IDS), Web Cache entre outros. A ação utilizada seria a de encaminhamento para a porta em que o Middlebox se encontra. A linha **14** da figura 4.9 demonstra o encaminhamento de pacotes oriundos do site `baixaki.com.br` para uma aplicação de anti-vírus.

Capítulo 5

Discussão

Neste capítulo serão apresentadas questões pontuais e como elas se relacionam com esta dissertação. Estas questões geram discussão sobre o porque da solução adotada. Para esclarecer melhor as escolhas tomadas será tratado cada ponto e o porque da decisão. Os pontos tratados são as outras formas de obtenção de nome de domínio, a utilização dos Middleboxes, instalação do interpretador DNS no controlador, servidores que utilizam múltiplos servidores web *(Shared Web Hosting)*, migração de conexões TCP, a utilização de regra de controle de acesso, e por último, o OpenFlow Extensible Match.

5.1 Outras Formas de se Obter o Nome de Domínio

Existem outras formas que podem ser implementadas para obter nomes de domínio. Uma das possíveis soluções é através do DNS Reverso. É possível obter a relação <IP, Nome de Domínio> interceptando a mensagem de DNS reverso e salvando-a na tabela. Importante ressaltar que, nem todos os Domínios possuem o recurso de DNS reverso habilitado.

Outra forma é o *parsing* de pacotes HTTP. Pacotes HTTP contêm o campo Host. A partir do campo Host é possível retirar o URL. Esta implementação é interessante por ser mais abrangente. Entretanto, possui como problema a fragmentação do protocolo IP. Apenas o primeiro pacote possui o campo Host. Assim os outros pacotes precisam de uma forma de atribuir o URL acessado.

5.2 Middleboxes

Os sistemas de rede atuais utilizam Middleboxes para lidar com pacotes da camada de aplicação. Middleboxes são dispositivos de propósito único. Podem ser implementados em hardware ou software. Os pontos negativos da utilização de um Middlebox são: a dependência do fabricante, seu alto custo, e a necessidade de mão de obra especializada. Exemplos de Middleboxes são: balanceadores de carga para servidores web, proxies, firewalls entre outros [Bezahaf et al., 2013].

Nossa proposta pode substituir ou até trabalhar em conjunto com Middleboxes. Aplicações que necessitam de salvar o estado podem ser implementadas no controlador, ou encaminhadas para os Middleboxes corretos. Esta última técnica é chamada de *traffic steering*, e é utilizada para melhorar o tráfego dentro do ambiente de redes.

5.3 Abordagem Utilizando o Controlador

O padrão OpenFlow já possui a capacidade de realizar regras de casamento baseadas em endereço IP. A razão de não utilizar essa abordagem é a possibilidade de um domínio utilizar mais de um endereço IP, portanto é uma tarefa difícil criar regras para todos endereços IPs para um único domínio. Além disso, existe a possibilidade de um domínio trocar o endereço IP utilizado. Desta forma, regras baseadas apenas em endereços IPs podem não funcionar como esperado.

Outra forma de implementar uma solução similar à nossa proposta é possível através do *parsing* dos pacotes de aplicação no controlador. Basta o controlador possuir uma lista dos domínios e a ação desejada para cada domínio. O controlador fica em cargo de extrair dos pacotes os Nomes de Domínio e realizar a busca na lista de domínios. Caso ocorra casamento na lista para aquele domínio, o controlador instala a regra de casamento no switch utilizando o endereço IP do domínio. Este método pode aumentar o tráfego de rede, devido a necessidade de troca de mensagens entre controlador e switch, e aumentar a latência da aplicação. Este custo adicional é devido ao tempo gasto pelo controlador para processar o pacote.

5.4 Múltiplos Servidores Web

Um problema relacionado é o Shared Web Hosting. Essa técnica consiste em hospedar vários web sites utilizando o mesmo endereço IP. A atual proposta é capaz de dizer se o IP está alocado para mais de um nome de domínio e avisar o usuário, deixando com ele a decisão de como proceder. Outro ponto importante é a incompatibilidade do Shared

Web Hosting com HTTPS. Neste caso, é utilizado apenas um certificado digital para todos os domínios hospedados naquele servidor, o que torna o sistema menos seguro.

5.5 Migração de conexões TCP

A implementação de uma aplicação de balanceamento de carga, ou uma regra de encaminhamento que troque as portas em que um fluxo TCP já foi estabelecido pode causar problemas. Portanto, uma forma de migrar conexões TCP sem nenhum dano para o usuário precisa ser implementada. Sistemas como SockMI [Bernaschi et al., 2007] permitem a migração de fluxos de uma maneira simples, sem modificações, apenas utilizando sistemas Linux. Para solucionar este problema, uma abordagem similar pode ser adotada em nosso projeto.

5.6 Regras ACL

Regras ACL (**A**ccess **C**ontrol **L**ist) são um mecanismo de segurança. ACL funciona da seguinte forma, o administrador escolhe a faixa de portas e endereços IPs para permitir ou negar acesso. É possível bloquear uma faixa de IPs para evitar o acesso a alguns servidores. Assim como nossa proposta regras ACL utilizando faixas de IP CIDR podem economizar a quantidade de regras na tabela do switch. Contudo, esta abordagem não é confiável em caso de mudanças de endereços IPs. Sobretudo, este mecanismo pode ser incorporado à nossa solução.

5.7 OpenFlow OXM TLV

A versão 1.2 do protocolo OpenFlow apresenta o **O**penFlow e**X**tensible **M**atch (OXM). O OXM permite que a estrutura de casamento, antes rígida e pré-definida, seja mais flexível e que receba novos campos de casamento com maior facilidade. Este recurso auxilia a experimentação e a inclusão de novos campos à estrutura de casamento do protocolo OpenFlow.

TLV ou ***T**ype **L**ength **V**alue* é uma forma utilizada de codificar a estrutura de casamento do protocolo OpenFlow. TLV é uma tupla que possui 3 valores. O primeiro valor, *type*, representa qual cabeçalho das camadas L2 a L4 deve ser casado. O segundo valor, *length*, é utilizado para indicar o tamanho do terceiro campo. O terceiro campo, *value*, contém as informações que serão utilizadas para o casamento.

A codificação OXM TLV trouxe para o protocolo uma maior organização dos campos utilizados no casamento. É possível diferenciar os campos utilizados das classes pertencentes, sejam eles campos obrigatórios, opcionais, ou de experimentação. Contudo, sua implementação ainda não permite a possibilidade de casar dados provenientes do *payload* dos pacotes. O nosso projeto pode ser implementado utilizando o OpenFlow OXM TLV, utilizando uma versão do OpenFlow igual ou maior que a 1.2. Desta forma, a inclusão de novos campos é facilitada.

5.8 Implementação em Hardware

Nosso projeto é possível de ser implementado em Hardware. Uma das possíveis formas é utilizando o conceito apresentado em [Naous et al., 2008]. Este artigo demonstra como é realizada a implementação do switch OpenFlow na plataforma NetFPGA. Outra forma de implementação, desta vez em switches comuns, a nossa estrutura adicionada pode ser alocada na memória genérica do switch. Desta forma, são realizados apenas casamentos exatos na tabela <IP, Nome de Domínio>.

Capítulo 6

Avaliação

Neste capítulo é apresentada a avaliação do sistema DN+OpenFlow. São avaliados o novo switch e controlador, o DN+OpenFlow e o DN+POX respectivamente. A avaliação do switch consiste em medir o custo adicional gerado pela proposta. São medidos os custos adicionais de CPU, Memória, Latência e Vazão. Como foram adicionadas novas estruturas de dados e uma série de instruções ao pipeline do switch, é interessante medir o *overhead* gerado por essas alterações. Cada recurso avaliado possui um experimento específico, detalhado em sua seção correspondente.

A primeira seção, 6.1, apresenta o ambiente de testes utilizado. A seção 6.2 apresenta o Trace criado para avaliar o DN+OpenFlow. A seção 6.3 realiza a comparação entre regras utilizando endereços IP versus regras utilizando nomes de domínio. A seção 6.4 apresenta os experimentos para avaliar o custo adicional *(overhead)* obtido em CPU e Memória. A seção 6.5 apresenta os experimentos realizados para latência. A seção 6.6 apresenta os resultados obtidos para os experimentos de vazão. A seção 6.7 compara o grau de abstração do DN+OpenFlow em relação ao OpenFlow convencional. A seção 6.8 apresenta como utilizar os contadores estatísticos do DN+OpenFlow para recuperar dados sobre os nomes de domínio acessados. Por último a seção 6.9 provê um exemplo prático da utilização do DN+OpenFlow, o exemplo demonstrado é o bloqueio do serviço de mensagens WhatsApp.

6.1 Ambiente de Testes

DN+OpenFlow foi testado utilizando o Mininet [1] [Handigol et al., 2012; Lantz et al., 2010]. Todo o código foi executado na máquina virtual provida pelo Mininet [2]. O

[1] http://mininet.org

[2] https://github.com/mininet/mininet/wiki/Mininet-VM-Images

virtualizador utilizado foi o VMware Player, versão gratuita para o sistema Linux. O computador que executa a VM é uma estação de trabalho com processador Core i7 8 núcleos e 8 GB de memória RAM. A VM foi configurada com 2 núcleos de processamento e 2 GB de RAM, utilizando o sistema operacional Linux Ubuntu 14.04 LTS e kernel 3.13.

A topologia utilizada nos experimentos é descrita na figura 6.1. Esta topologia consiste em 2 hosts (**H1** e **H2**) sendo executados dentro do ambiente do Mininet. Foi utilizado um switch **S1** (DN+OpenFlow ou OpenFlow), 1 controlador **C0** (DN+Pox ou Pox executando o módulo `l2_learning`) e por último o depurador DPCTL (DN+DPCTL ou DPCTL padrão). Na máquina virtual foi instalado o servidor DNS BIND representado pela letra **B** na figura 6.1. Este servidor é utilizado para realizar testes locais e adicionar os nomes de domínio criados para o experimento. Os nomes de domínio criados foram `h1.dnopenfow.com/h2.dnopenflow.com` representado respectivamente os hosts h1 e h2.

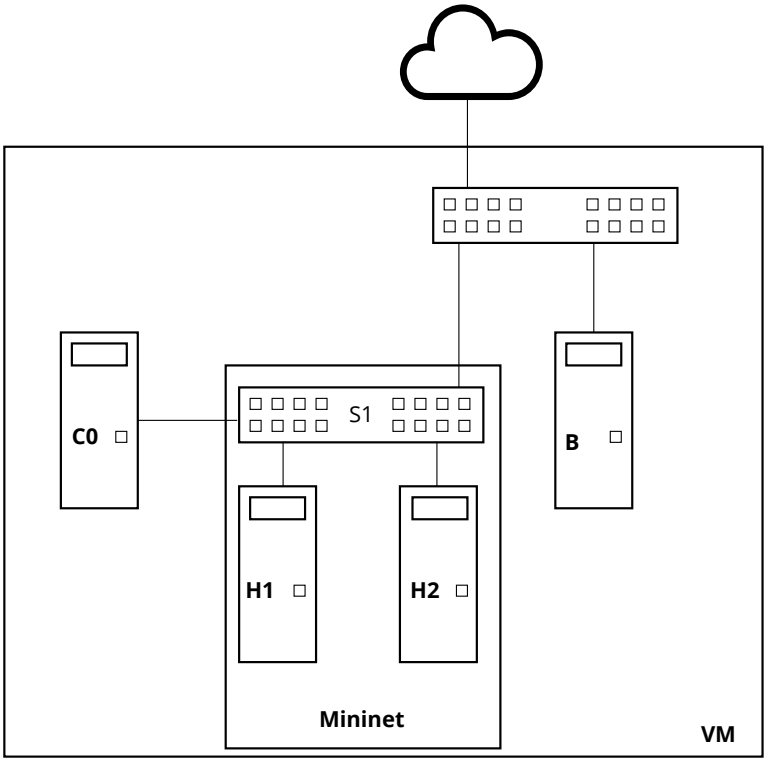


Figura 6.1. Topologia utilizada para realização dos experimentos

Algumas máquinas dos laboratórios de pesquisa Lecom e Winet foram utilizadas em alguns testes. O intuito foi variar a quantidade de *hops* (saltos) que o cliente está do servidor e ver o quanto isso afeta nos testes. São delimitados na figura 6.1 os limites dos

ambientes do Mininet e da Máquina Virtual-VM. A nuvem representa a rede externa, podendo ser a própria rede do Lecom ou Winet, ou a Internet.

6.2 *Trace* Utilizado

Foram coletados tráfego da porta 53 (DNS) protocolo UDP e porta 80 (HTTP) protocolo TCP do laboratório de pesquisa **Win**et [3], totalizando 5 GB de dados. Estes dados foram coletados durante 1 dia de acesso do laboratório. O trace conta com 50 IPs internos utilizados. A partir desses dados montamos um trace formado para testar o DN+OpenFlow. O *trace* foi divido para poder ser aplicado em várias ocasiões de testes. A tabela 6.1 apresenta o tamanho de cada trace, a quantidade de pacotes, e a quantidade de pacotes DNS e HTTP.

Tabela 6.1. Traces Utilizados

Trace	#Pacotes	#Pacotes DNS	#Pacotes HTTP
5GB	5854547	310173	5544374
1GB	1303513	155866	1147647
100MB	201003	53371	147632
50MB	101290	26775	74515
10MB	19040	5156	13884

A tabela 6.2 apresenta as top 15 requisições DNS salvas pelo *trace*. É importante ressaltar a quantidade de nomes de domínio de produtos de empresas conhecidas como Google e Facebook. Outro aspecto importante é que das 15 requisições mais solicitadas apenas uma possui 1 endereço IP associado. Todas as outras requisições DNS possuem mais de 1 endereço IP associado. Ao analisar as top 100 requisições apenas 13 nomes de domínio possuem apenas 1 endereço IP associado.

No total foram coletados 7035 nomes de domínios diferentes. Destes 7035 nomes de domínio apenas 23% possuem apenas 1 endereço IP associado. A tabela 6.3 apresenta a quantidade de nomes de domínio que possuem 1, 2, 3 e até 17 endereços IPs associados.

A Figura 6.2 ilustra a distribuição cumulativa (CDF) de endereços IPs por nomes de domínio. Cerca de 23% dos nomes de domínio possuem apenas um endereço IP. Os 77% restantes possuem pelo menos dois endereços IPs. Esta informação é útil para afirmar a utilização de regras utilizando Nomes de Domínio.

[3] www.winet.dcc.ufmg.br

Tabela 6.2. Top 15 requisições DNS

#	Nome de Domínio	# Requisições	# IPs
1	daisy.ubuntu.com	11176	6
2	mail.google.com	3367	4
3	clients6.google.com	3360	4
4	play.google.com	2972	15
5	www.facebook.com	2038	3
6	plus.google.com	1816	15
7	0.client-channel.google.com	1755	1
8	www.google.com	1588	5
9	star.c10r.facebook.com	1488	2
10	centos.ufms.br	1206	2
11	clients4.google.com	1171	14
12	3-edge-chat.facebook.com	1120	3
13	safebrowsing-cache.google.com	1008	15
14	googleads.g.doubleclick.net	946	7
15	geoip.fedoraproject.org	913	9

Tabela 6.3. Relação Nome de Domínio versus Quantidade de IPs

#IPs	# Nomes de Domínio	% no *Trace*
1	1620	23
2	709	10
3	682	10
4	818	11
5	1229	17
6	459	6
7	289	4
8	354	5
9	228	3
10	177	2
11	147	2
12	173	2
13	62	1
14	8	0.1
15	48	0.7
16	21	0.3
17	11	0.1

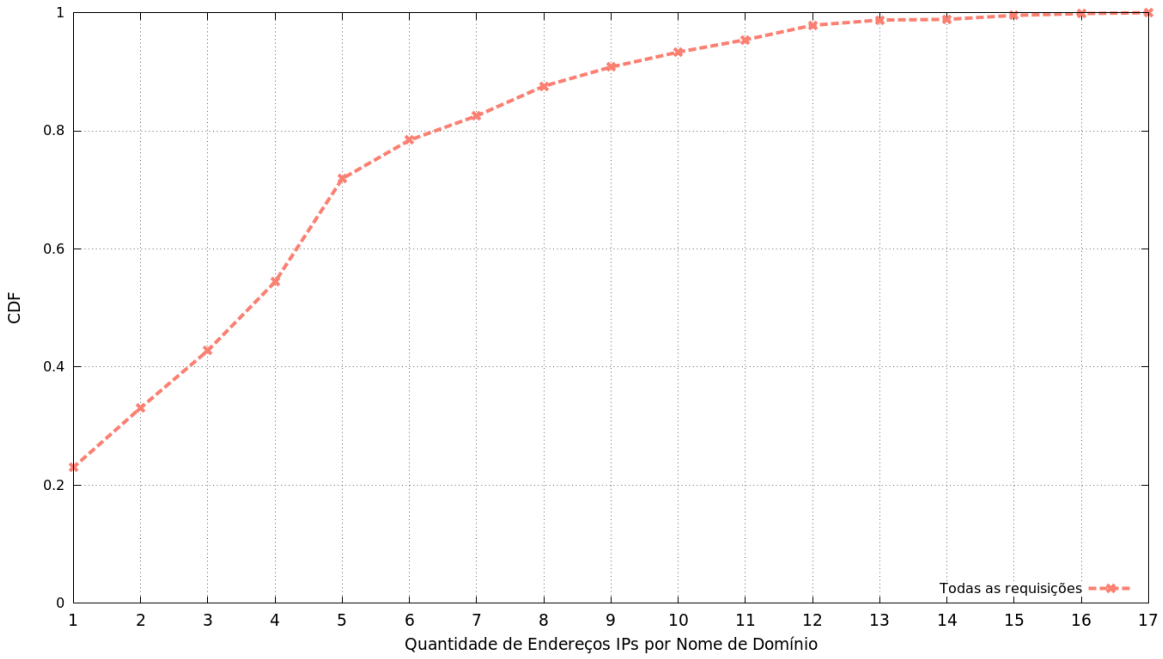


Figura 6.2. CDF Endereços IPs por Nomes de Domínio

6.3 Regras Utilizando IPs x Regras Utilizando Nomes de Domínio

Para criar regras para um domínio utilizando endereços IPs é necessária uma regra para cada endereço IP. Essas regras são conhecidas como ACL, regras de controle de acesso. Contudo, os dispositivos de rede possuem uma quantidade máxima de regras suportadas. Uma forma de economizar a quantidade de regras na tabela do dispositivo é utilizar regras por nome de domínio. Assim, todas as regras para um único domínio podem ser substituídas por apenas uma regra por domínio.

A figura 6.3 ilustra a relação entre endereço IP e nos de domínio por trace. Cada resposta DNS possui endereços IPs para o nome de domínio solicitado. As barras vermelhas demonstram a quantidade de nomes de domínio e as barras azuis a quantidade de endereços IPs. A linha laranja representa a quantidade de regras do tipo ACL suportada pelo switch de agregação Cisco Nexus 6000 [4] (4000 regras), e a linha verde representa a quantidade de regras ACL suportadas pelo switch topo de rack Dell S4048 [5] (1024 regras). Foram escolhidos dois switches com capacidades diferentes. O switch Dell topo de rack possui uma tabela menor, de acordo com a sua aplicação.

[4]http://www.cisco.com/c/en/us/products/collateral/switches/nexus-6000-series-switches/datasheet-c78-732277.html

[5]http://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Dell-Networking-S4048-ON-Spec-Sheet.pdf

O outro switch, de agregação, o Cisco, é um switch que lida com uma quantidade de fluxos maior, logo, possui uma tabela maior.

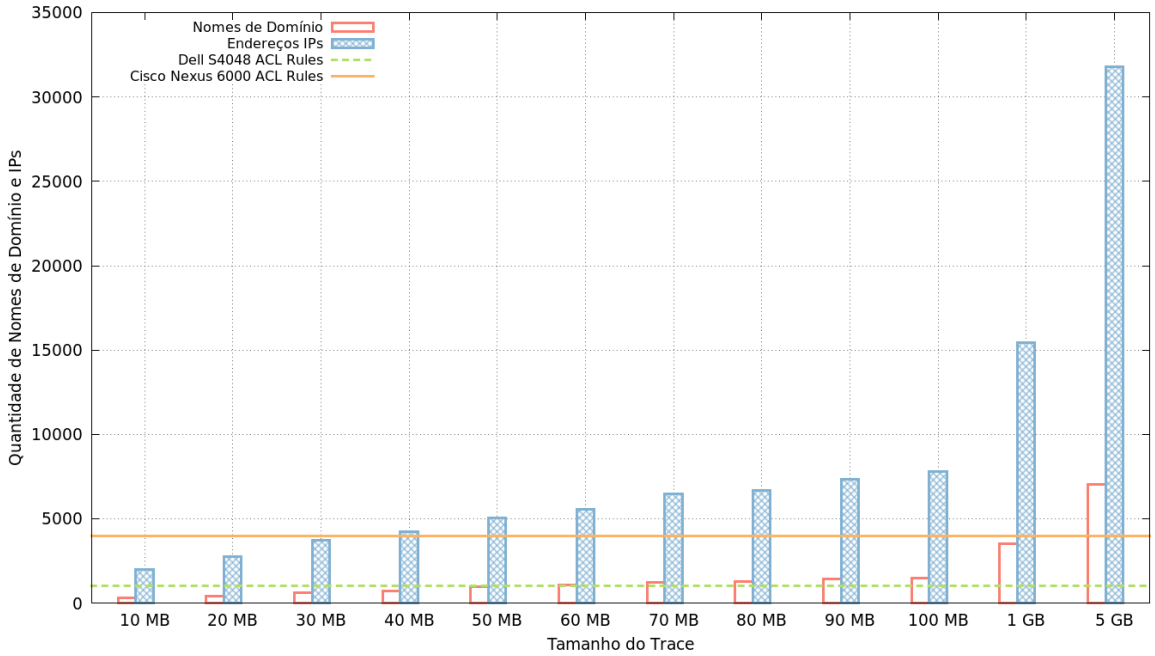


Figura 6.3. Quantidade de endereços IPs e Nomes de Domínio por trace

Ao utilizar o DN+OpenFlow é possível diminuir a quantidade de regras na tabela de fluxos do switch. Para isso é necessário incluir uma regra que agregue os endereços IPs em apenas um nome de domínio. Desta forma os endereços IPs de origem e destino não serão utilizados para casamento, apenas os nomes de domínio de origem e destino. Assim, os vários endereços IPs existentes para um site serão agregados em uma regra só, sendo possível consultar a quantidade de bytes e pacotes que casaram naquela regra. Esta funcionalidade será descrita mais detalhadamente na seção 6.8.

A figura 6.4 ilustra o tempo de instalação de regras para os 15 domínios mais acessados em nosso trace, apresentados na tabela 6.2. O eixo x demonstra a quantidade de nomes de domínio e o eixo y o tempo de instalação de regras em segundos. A linha vermelha apresenta a solução DN+OpenFlow, e a linha vermelha representa a implementação original do OpenFlow. O DN+OpenFlow precisa de menor tempo para instalar as regras que o OpenFlow original, devido a quantidade menor de regras necessárias. A implementação padrão do OpenFlow possui valores variados devido à quantidade de endereços IP variar de acordo com o domínio requisitado. O tempo de inserção gasto pelo DN+OpenFlow pode chegar a ser 5 vezes menor em comparação com a implementação padrão do OpenFlow.

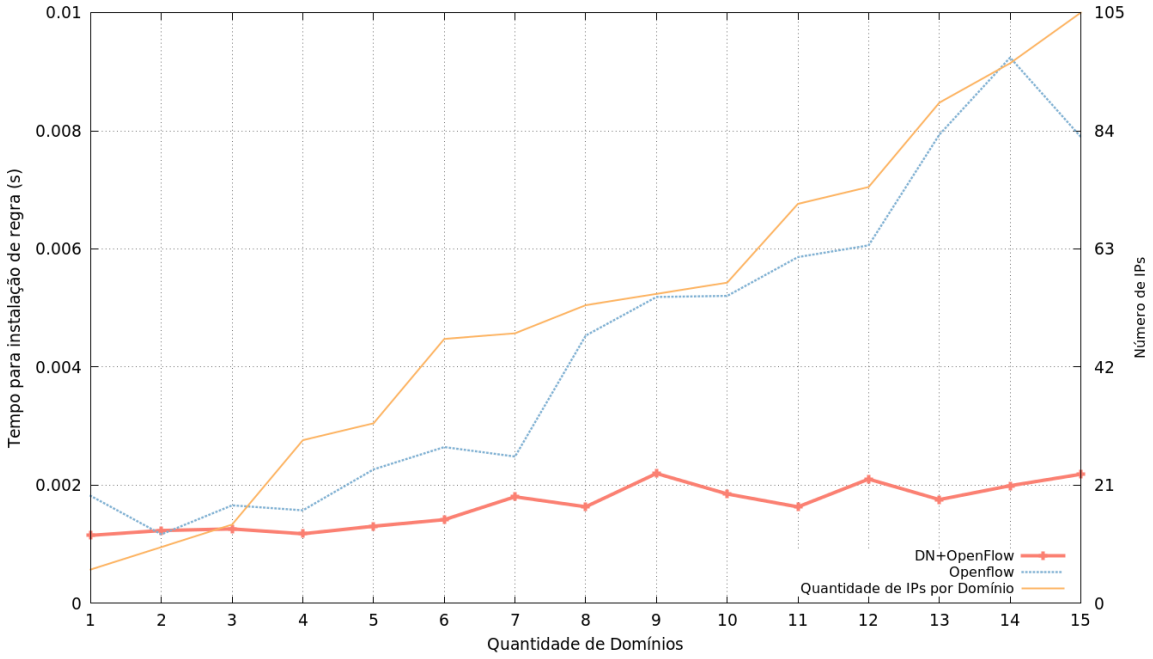


Figura 6.4. Tempo de instalação de regras para o DN+OpenFlow e OpenFlow

6.4 Custo adicional de CPU e Memória

A arquitetura DN+OpenFlow possui as seguintes instruções que podem impactar em tempo de CPU: Interpretação de pacotes DNS de resposta, operações da tabela <Endereço IP, Nome de Domínio> (inserção, remoção, busca), atribuição de nome de domínio aos pacotes, e a função de comparação de nomes de domínio para realização do casamento.

O DN+OpenFlow adiciona as seguintes modificações que podem impactar o uso de memória: Adição às estruturas de fluxos e casamento os campos `DN_SRC` e `DN_DST` e a criação da tabela <IP, Nome de Domínio>. Para avaliar o custo adicional gerado pelas modificações, foi utilizado o trace coletado. Foram medidos os consumos de CPU e Memória e comparadas com a implementação padrão do OpenFlow e do Pox.

O consumo de CPU e memória foram medidos utilizando o comando "top". O trace foi replicado utilizando a ferramenta *tcpreplay* [6]. Para evitar que o switch descarte os pacotes, os endereços MAC e IP de destino foram reescritos utilizando o *tcprewrite*. Foram utilizados os endereços adotados pela topologia do Mininet. Assim, podemos testar switch e controlador. Os pacotes do trace passam pelo interpretador do plano de dados do switch OpenFlow. O controlador, por sua vez realiza a inserção dos fluxos na tabela do switch.

[6]http://tcpreplay.synfin.net/tcpreplay.html

Foi utilizado o trace de 1GB, totalizando 1.303.513 pacotes. Foi escolhido este trace porque os traces de tamanhos menores não causaram alterações significativas no consumo de CPU e Memória. O trace foi replicado à velocidade de 1Mbps, utilizando apenas 1 interface de rede de cada host. A origem foi o host h1 e o destino o host h2. A tabela 6.4 apresenta o resultado do teste realizado. O switch DN+OpenFlow foi 6,6% (cerca de 6 segundos) mais lento que a implementação padrão utilizada como base.

Tabela 6.4. Pacotes por segundo e tempo de execução do trace de 1 GB

Switch	Pacotes por segundo (pps)	Tempo Gasto (s)
DN+OpenFlow	13299,80	98,01
OpenFlow	14177,87	91,94

A figura 6.5 apresenta todos os valores de consumo de CPU coletados para o teste realizado. A implementação DN+OpenFlow possui valores maiores devido o interpretador de pacotes DNS utilizado e as funções necessárias para operar a tabela <IP, Nome de Domínio>.

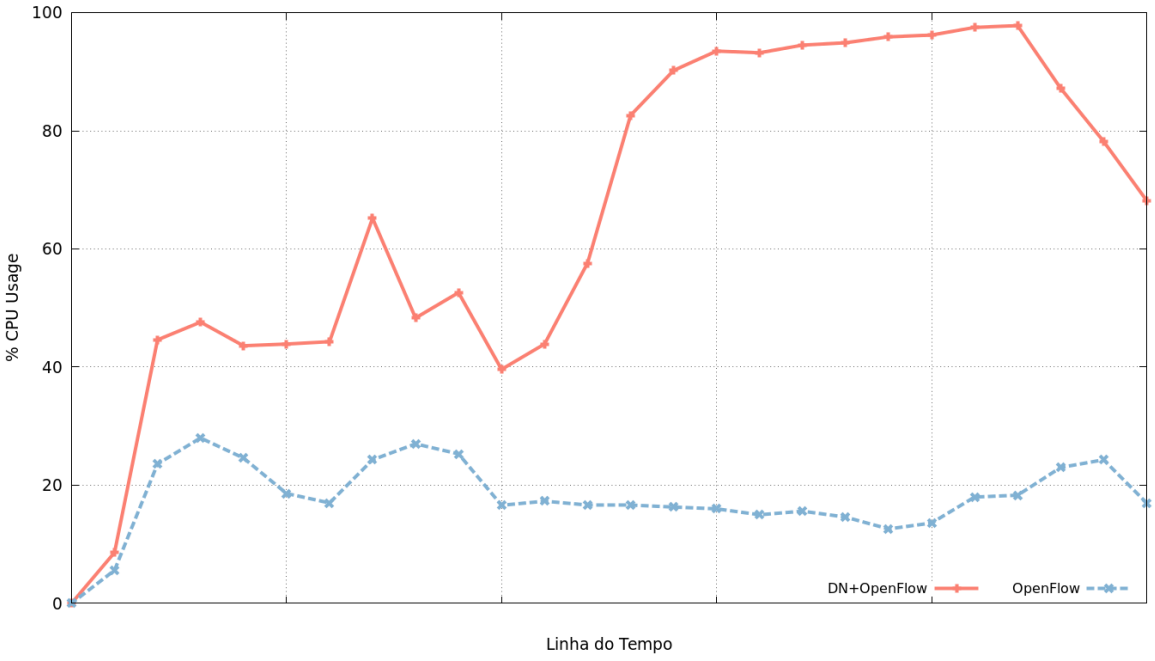


Figura 6.5. Consumo de CPU dos switches DN+OpenFlow e OpenFlow

O fato de o trace possuir apenas pacotes DNS e HTTP ao mesmo tempo sobrecarrega e testa as funcionalidades adicionadas para o funcionamento da solução. A tabela 6.5 apresenta os valores obtidos para consumo médio de CPU (%) e Memória.

O DN+OpenFlow consumiu em média 3,7 vezes mais CPU do que a versão padrão do switch OpenFlow. O consumo de memória permaneceu similar para todas aplicações.

Tabela 6.5. Consumo de Memória e CPU para os Switches DN+OpenFlow e OpenFlow

Switch	%CPU	Intervalo de Confiança	%RAM	Intervalo de Confiança
DN+OpenFlow	65,75	±10,89	0,18	±0,02
OpenFlow	17,9	±2,40	0,18	±0,02

A figura 6.6 apresenta todos os valores coletados para o consumo de CPU nos controladores DN+Pox e Pox. No início do experimento os dois controladores apresentam comportamento similar. Contudo, quando o switch DN+OpenFlow é sobrecarregado, os pacotes são atrasados e demoram para serem repassados ao controlador. Assim, a queda de processamento no controlador DN+Pox.

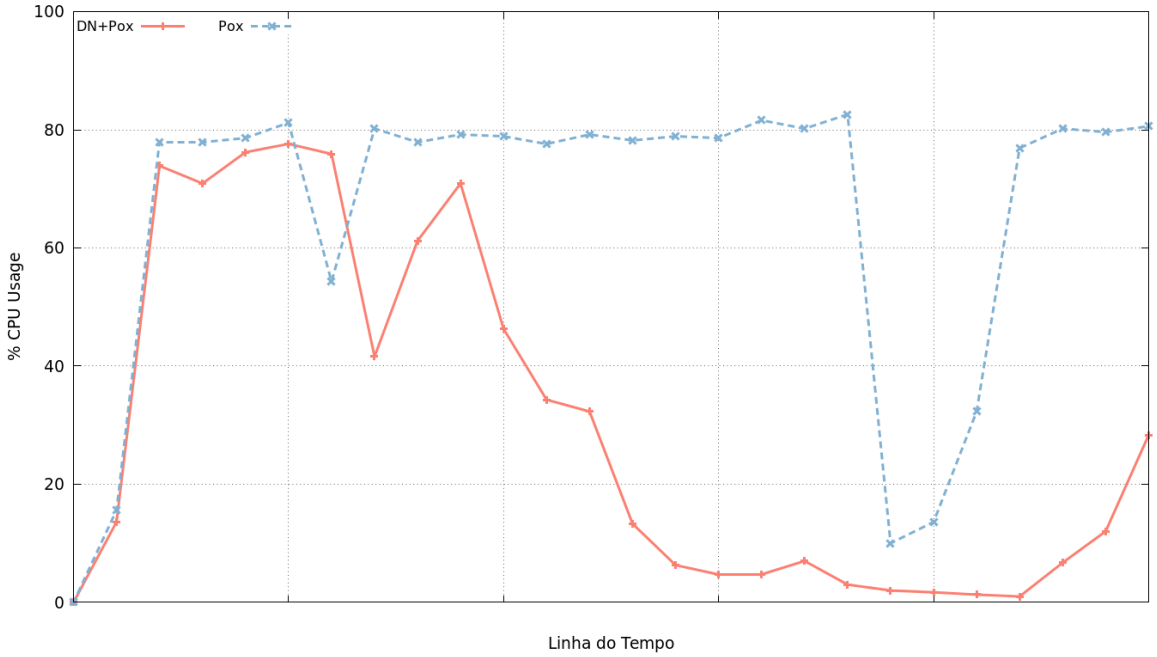


Figura 6.6. Consumo de CPU dos controladores DN+Pox e Pox

A tabela 6.6 apresenta os valores consumidos de CPU e memória para os controladores DN+Pox e Pox. O consumo de CPU do controlador DN+Pox foi 2,23 vezes menor em média em comparação com a implementação padrão do Pox. Isto se deve a grande carga de operações adicionadas ao plano de dados do switch DN+OpenFlow, reduzindo o tratamento dos pacotes e desafogando o controlador. O consumo de memória foi em média 20% maior em comparação à implementação padrão do Pox. Isto

se deve a necessidade de realizar o parser dos pacotes DNS e armazenar o dicionário com a relação <IP, Nome de Domínio> no controlador.

Tabela 6.6. Consumo de Memória e CPU para os Controladores DN+Pox e Pox

Controlador	%CPU	Intervalo de Confiança	%RAM	Intervalo de Confiança
DN+Pox	29,49	±11,36	0,96	±0,02
Pox	65,84	±10,20	0,8	0

A figura 6.7 apresenta os valores agregados do consumo de CPU para cada solução. O DN+OpenFlow aliado ao DN+Pox representam a solução proposta nesta dissertação. Juntos consumiram cerca de 95% da CPU da VM durante o experimento. Já a solução base utilizada para comparação utilizou cerca de 85%. DN+OpenFlow apresentou cerca de 10% de consumo a mais. Vale ressaltar que isto é devido às várias funcionalidades adicionadas ao switch e controlador.

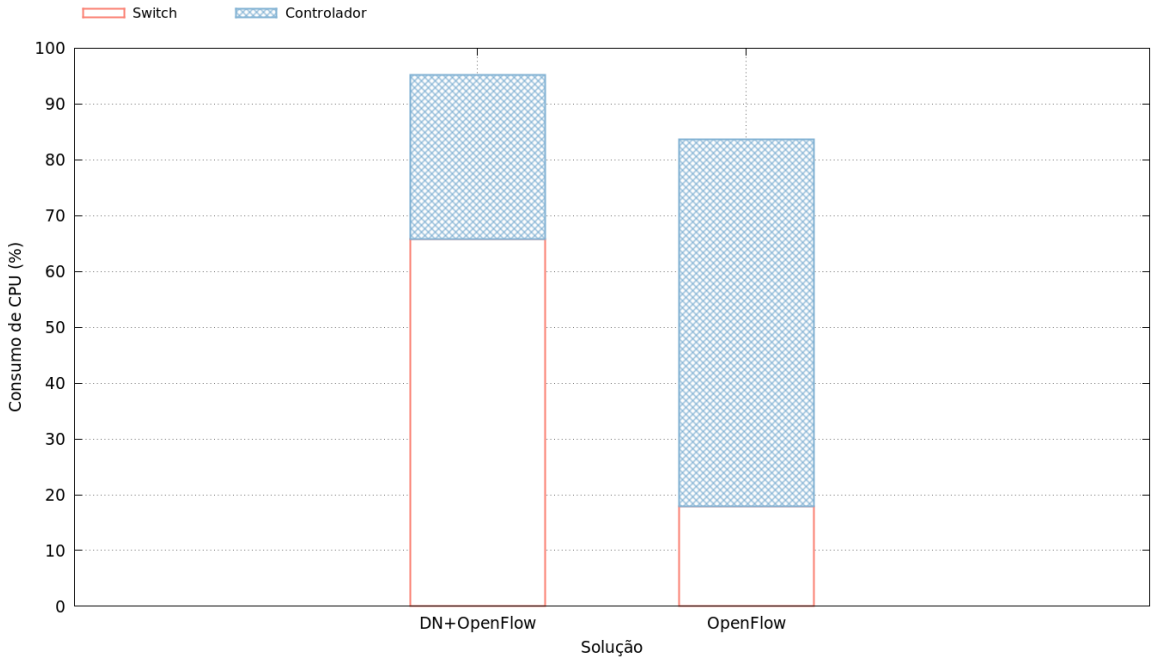


Figura 6.7. Valor agregado de CPU para cada solução

A figura 6.8 apresenta os valores de memória gastos para todos os programas avaliados nessa seção. Os valores encontrados comprovam que memória não é o maior custo adicional encontrado nesta dissertação. Além do que as estruturas adicionadas criadas (<Tabela IP,Nome de Domínio>) não comprometem o desempenho global da aplicação.

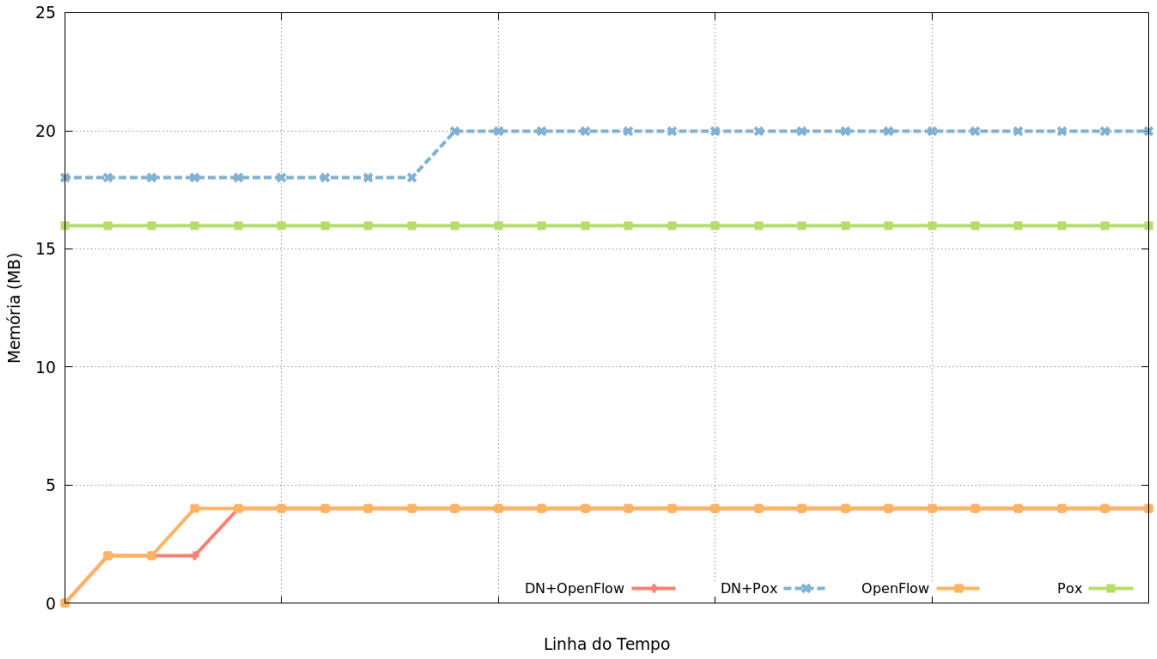


Figura 6.8. Utilização de memória para os programas testados

6.5 Latência

Para realizar os testes de latência foram utilizados os seguintes recursos. Dois hosts ligados ao switch, ambos com acesso externo à Internet. Na máquina virtual foi instalado um servidor DNS Bind. Neste servidor DNS foi armazenado o nome de domínio para os hosts locais. Os seguintes endereços foram assinalados `h1.dnopenflow.com` para o host 1 e `h2.dnopenflow.com` para o host 2. Assim, as requisições DNS realizadas em testes locais não sofrerão com possíveis problemas relacionados ao mundo externo.

O switch OpenFlow utiliza o seguinte módulo Pox para funcionar: `forwarding.l2_learning`. Assim, o switch funciona no modo de aprendizado. À medida que os pacotes vão chegando ao switch são armazenados os endereços MAC e a porta relativa àquele endereço. Para comprovar esse funcionamento o gráfico representado na figura 6.9 foi criado.

A figura 6.9 representa um teste ping ao endereço `www.google.com.br`. O primeiro pacote possui latência maior que os demais devido ao comportamento explicado. O primeiro pacote ao chegar ao switch não encontra regra de casamento para aquele fluxo, logo o pacote é encaminhado ao controlador. O controlador por sua vez é responsável por determinar qual regra deve ser aplicada àquele fluxo e assim instalar a regra no switch. A mensagem `flow_mod` é responsável por instalar fluxos no switch.

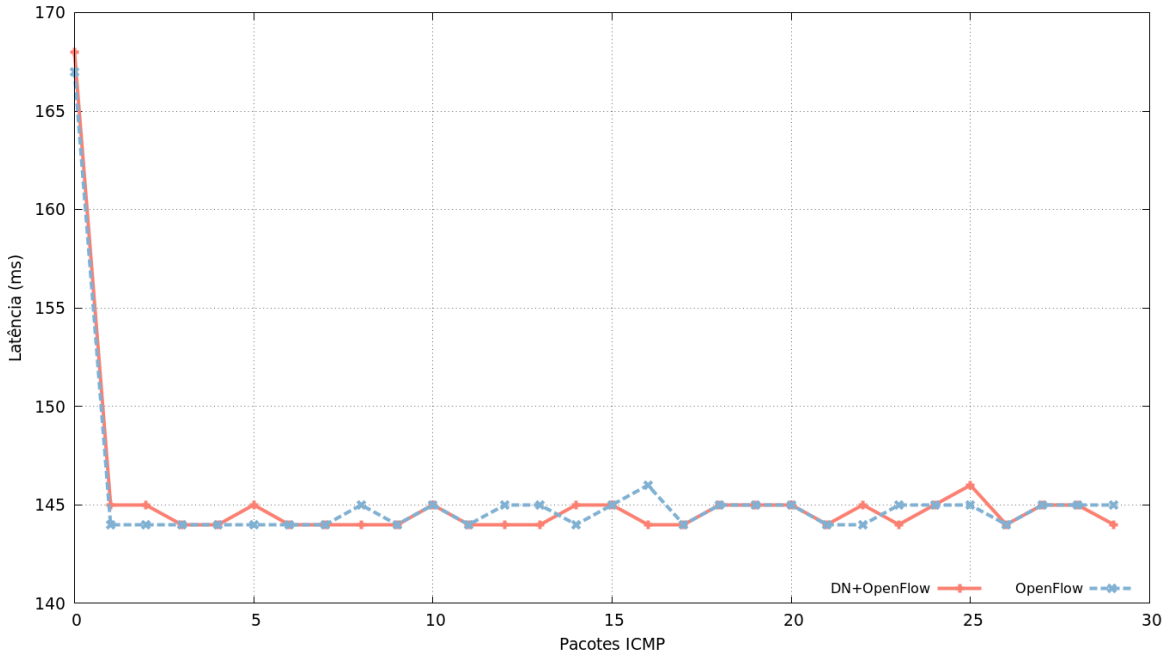


Figura 6.9. Latência www.google.com.br

Após a instalação da regra no switch, os demais pacotes casam com a regra e não precisam ser reenviados para o controlador. Desta forma, a latência é estabilizada. Neste teste os valores encontrados para ambas soluções foram similares. Logo, as alterações realizadas no DN+OpenFlow possuem impacto mínimo na latência. A tabela 6.7 possui valores para os outros nomes de domínio testados. Os valores obtidos para as duas soluções foram bem próximos. O número de hops é colocado na tabela para saber a distância que o servidor se encontra. A quantidade de hops foi obtida via *traceroute*.

Tabela 6.7. Resultados para latência para o DN+OpenFlow e OpenFlow

Nome de Domínio	DN+OpenFlow	OpenFlow	Hops
h2.dnopenflow.com	0,27 ± 0,16	0,28 ± 0,22	1
formiga.lecom.dcc.ufmg.br	0,81 ± 0,16	0,89 ± 0,13	3
www.winet.dcc.ufmg.br	0,84 ± 0,13	0,98 ± 0,21	4
www.ufmg.br	1,19 ± 0,11	1,44 ± 0,38	5
www.google.com.br	145,3 ± 1,55	145,3 ± 1,48	26

A figura 6.10 apresenta o gráfico comparativo entre os valores obtidos para latência medidos utilizando o *ping*. A latência não é alterada pelas mudanças realizadas na arquitetura do sistema DN+OpenFlow.

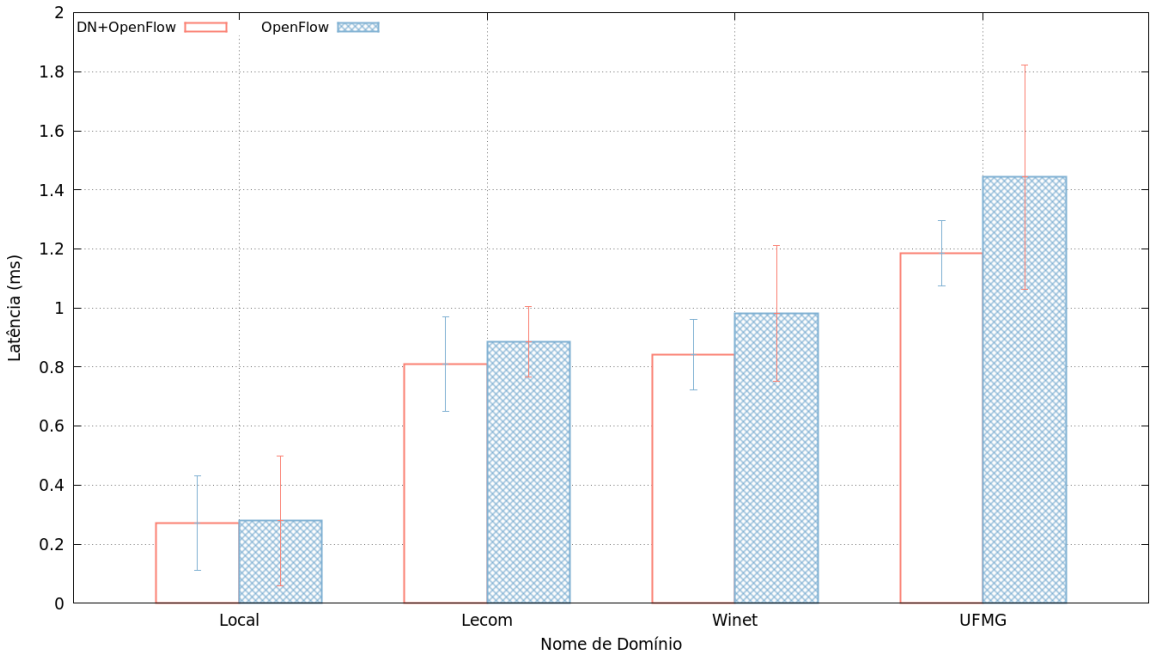


Figura 6.10. Teste de latência utilizando *ping*

6.6 Vazão

O primeiro teste de vazão realizado foi com o programa *iperf*. O primeiro teste foi feito dentro do ambiente do Mininet, utilizando o servidor DNS instalado na máquina virtual. Assim, as consultas DNS não sofrem com os problemas relacionados à rede externa. O controlador utilizado foi o Pox com o módulo `forwarding.l2_learning`

Tabela 6.8. Vazão utilizando Iperf (Mbps)

Nome de Domínio	DN+OpenFlow	OpenFlow
h2.dnopenflow.com	597,21 ± 4,47	685,29 ± 2,90
formiga.lecom.dcc.ufmg.br	94,89 ± 0,05	94,93 ± 0,03

A tabela 6.8 apresenta os valores obtidos para o teste utilizando o *iperf*. O protocolo testado foi o TCP. Para testes realizados localmente, solicitando o nome de domínio `h2.dnopenflow.com`, a vazão média obtida pelo DN+OpenFlow foi 14,74% menor que a vazão obtida pelo switch OpenFlow sem alterações. Foram transferidos 6,95 GB em 100 segundos pelo DN+OpenFlow e 7,98 GB em 100 segundos, cerca de 1 GB de diferença, correspondendo à cerca de 14% de *overhead*.

O teste feito para uma máquina física, a Formiga, localizada no laboratório Lecom (3 hops de distância) apresentou valores similares para ambos os testes. Isso é devido ao limite máximo da infraestrutura, no caso 100 Mbps. Assim, o gargalo maior

é da interface de rede e não o DN+OpenFlow. Contudo, vale ressaltar que os switches utilizados funcionam em nível de usuário. A figura 6.11 apresenta o gráfico para os testes realizados.

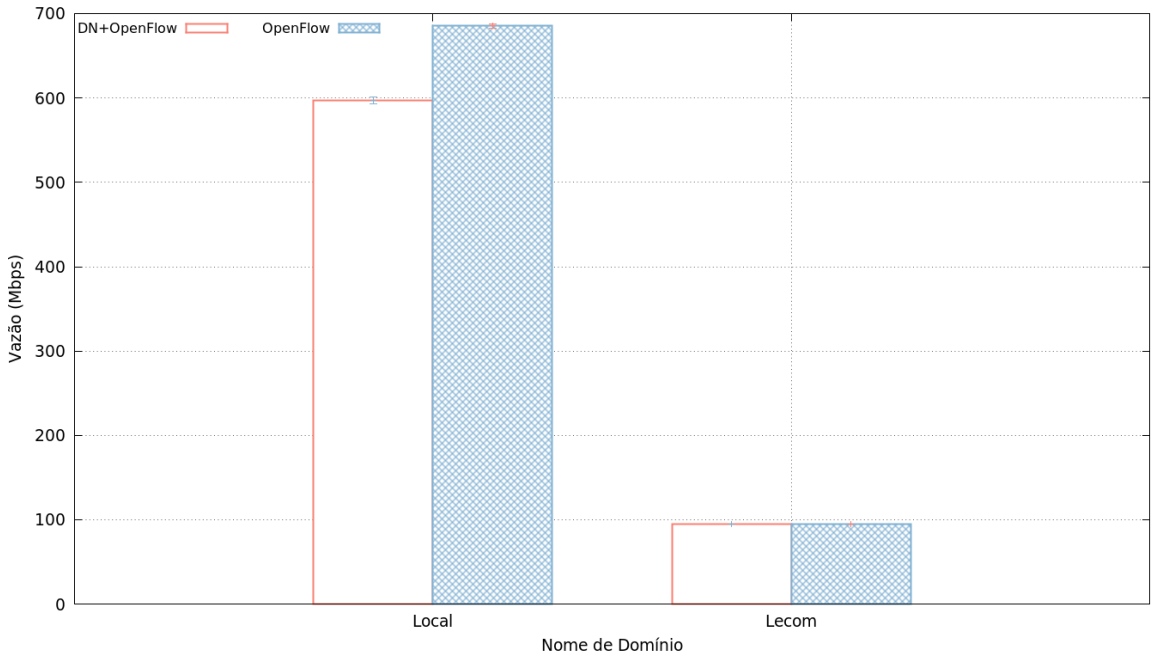


Figura 6.11. Vazão utilizando Iperf

A figura 6.12 apresenta a função cumulativa para os valores obtidos da vazão do experimento realizado localmente demonstrado na tabela 6.8. Para o DN+OpenFlow 80% dos valores da vazão se encontram acima de 600Mbps. Para o OpenFlow padrão 80% dos valores obtidos pela vazão se encontram entre 675Mbps e 700 Mbps.

Outra forma de avaliar a vazão é via o programa *wget*. Para isso foi montado um servidor HTTP em um dos hosts do Mininet. O servidor foi montado utilizando o SimpleHTTPServer [7] do Python. Foram utilizados arquivos de 10 MB, 100 MB, 1 GB para medir o tempo médio de *download* para cada solução. O nome de domínio dado ao servidor HTTP foi o `h2.dnopenflow.com`. A tabela 6.9 apresenta os valores obtidos para o teste.

Para pequenos tamanhos de arquivos as duas soluções possuem valores parecidos. Contudo, para arquivos maiores a solução DN+OpenFlow apresenta desempenho menor, chegando a ser 3 segundos mais lenta que a implementação padrão do protocolo OpenFlow. A figura 6.13 apresenta o gráfico para os resultados obtidos. Pode-se verificar a pequena diferença entre os valores.

[7] https://docs.python.org/2/library/simplehttpserver.html

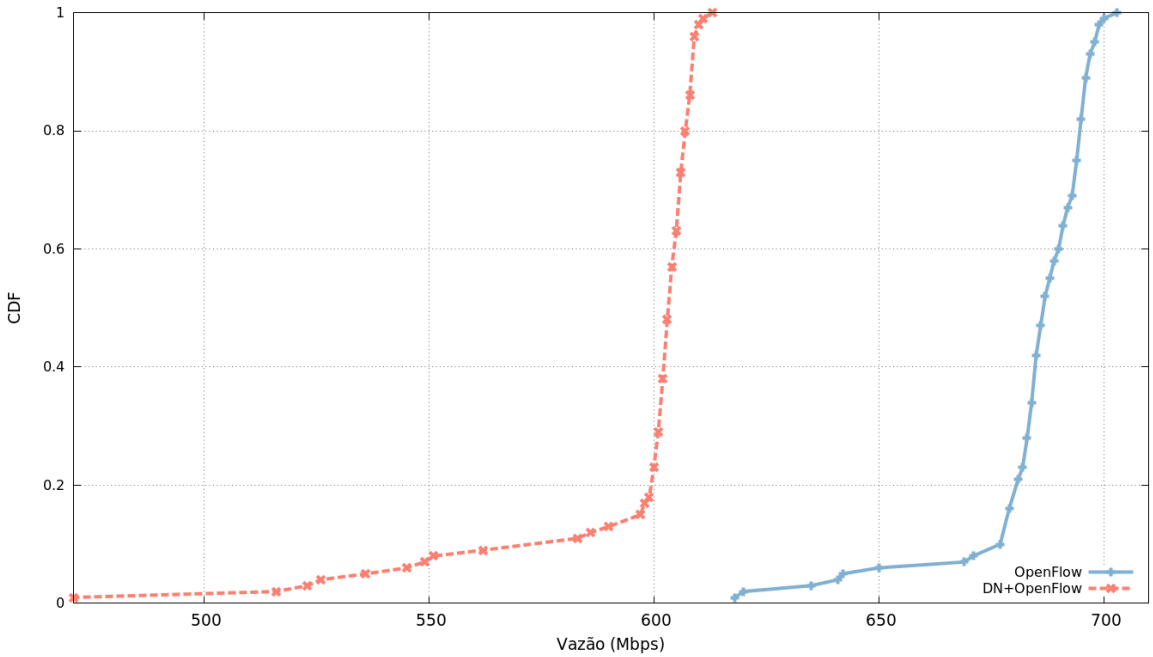


Figura 6.12. CDF Vazão utilizando Iperf

Tabela 6.9. Tempo (em segundos) gasto para download utilizando wget em servidor local

Tamanho do Arquivo	DN+OpenFlow(s)	OpenFlow(s)
10 MB	0,22 ± 0,03	0,18 ± 0,06
100 MB	1,89 ± 0,05	1,50 ± 0,08
1 GB	19,92 ± 0,75	16,69 ± 0,81

Além dos testes utilizando a rede local, também foram realizados testes utilizando páginas reais da Internet. A diferença entre os tempos coletados é bem pequena, chegando apenas a se diferenciar na última página baixada. A tabela 6.10 apresenta os valores obtidos para cada página. figura 6.14 apresenta o gráfico para o teste realizado. A diferença de valores obtidos é bem pequena quando comparado com a implementação padrão do protocolo OpenFlow.

Tabela 6.10. Tempo de Download para páginas da Internet

Nome de Domínio	DN+OpenFlow(s)	OpenFlow(s)
winet.dcc.ufmg.br	0,004 ± 0,00	0,004 ± 0,01
g1.globo.com	0,07 ± 0,00	0,07 ± 0,00
pt.wikipedia.org	0,41 ± 0,01	0,43 ± 0,01
yahoo.com	1,19 ± 0,05	1,19 ± 0,02
www.youtube.com.br	1,83 ± 0,04	1,69 ± 0,04

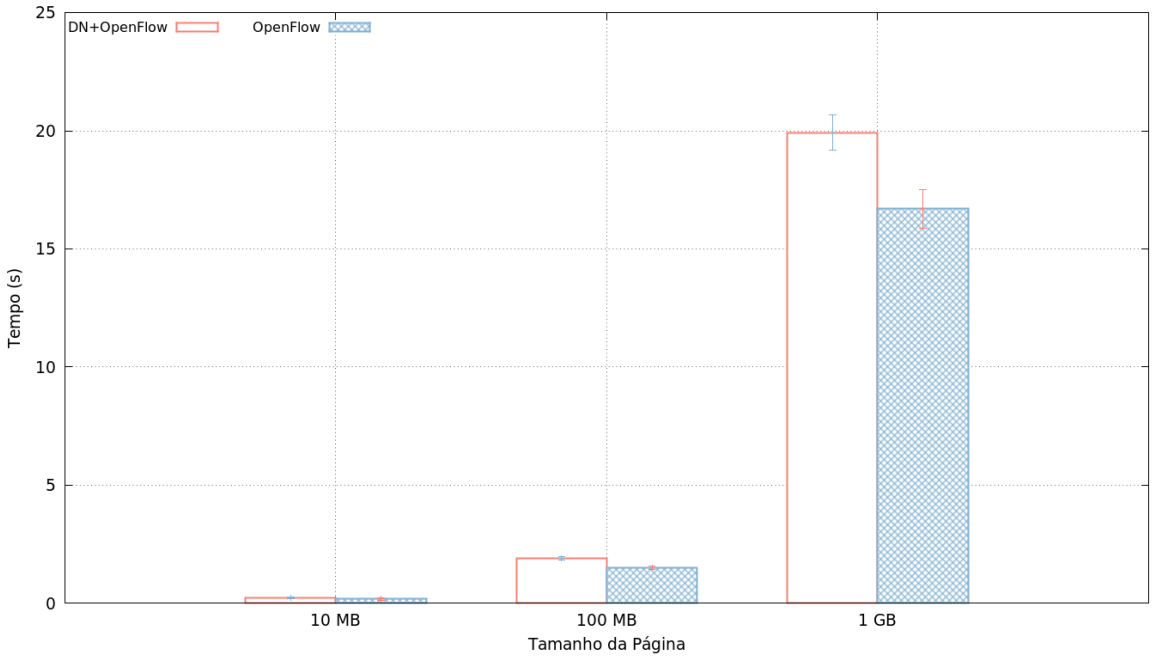


Figura 6.13. Tempo de Download medido em ambiente local do Mininet utilizando o wget

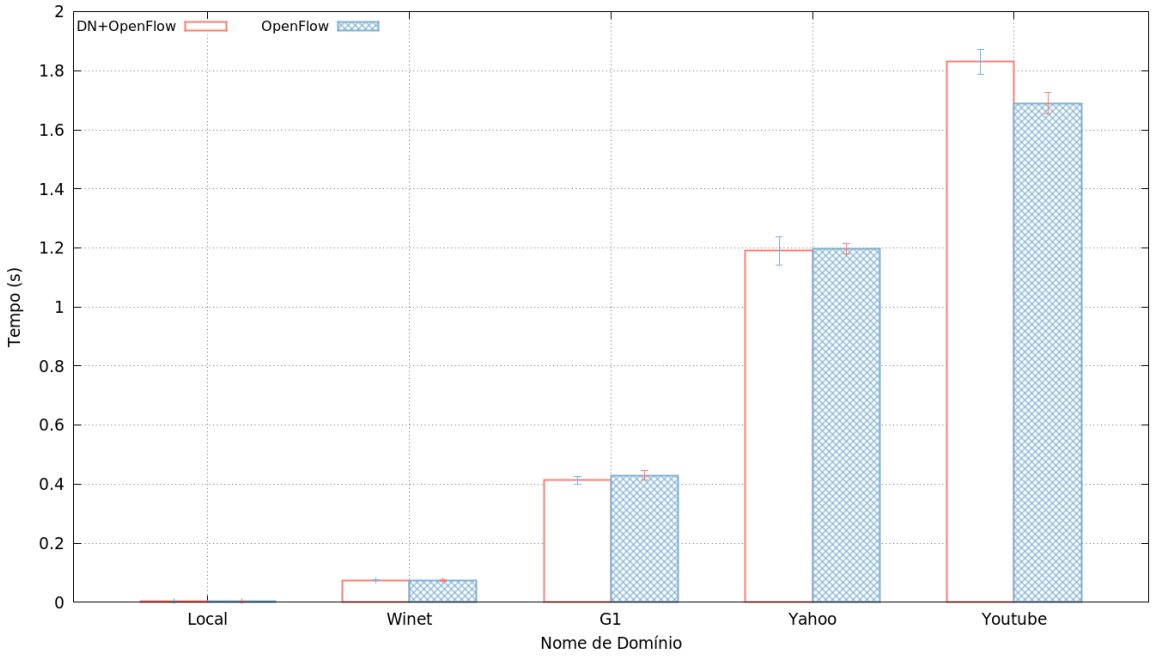


Figura 6.14. Tempo de Download(s) médio para páginas da Internet

6.7 Grau de Abstração

Foi comparada também a complexidade dos programas SDN. Para criar um Firewall utilizando nomes de domínio são necessárias 140 linhas de código utilizando o Pox. Enquanto isso, utilizando os campos adicionados pelo DN+OpenFlow, as mesmas funções podem ser realizadas com apenas 30 linhas de código. Além disso, o DN+OpenFlow reduz a quantidade de pacotes enviados ao controlador, pois o próprio switch já armazena a informação mais atualizada sobre um nome de domínio. Na implementação comum do Pox, o controlador precisaria receber cada pacote DNS de resposta para poder mapear a relação endereço IP, nome de domínio. Além da necessidade de manter a informação nome de domínio endereço IP atualizada. A figura 6.15 apresenta o gráfico comparando a quantidade de linhas de código necessárias para construir um Firewall, ou qualquer aplicação que crie regras utilizando nomes de domínio com o controlador Pox. As aplicações criadas podem ser conferidas no apêndice da dissertação.

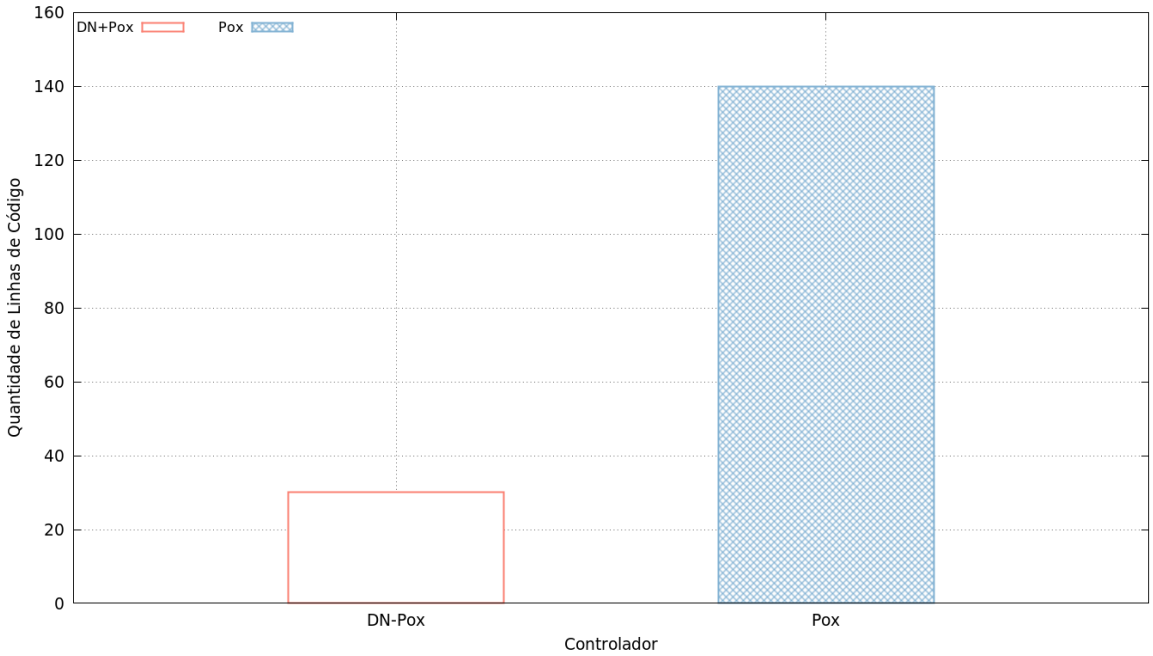


Figura 6.15. Linhas de código para cada programa SDN

6.8 Contadores estatísticos do OpenFlow

DN+OpenFlow também permite consultar dados sobre fluxos que possuem nomes de domínio associados. Após a regra de casamento ser inserida, é possível recuperar dados de cada fluxo. É possível recuperar por exemplo a quantidade de pacotes, número de

bytes, porta de entrada, ações, endereço IP de origem, endereço IP de destino e outros campos.

Para validar este teste foi adicionada uma regra de casamento para nomes de domínio de origem e destino para o site `www.dcc.ufmg.br`. Assim é possível obter estatísticas sobre essa regra em específico. Então, foi utilizado o comando "wget"para fazer o download do site do DCC. Após isso, foi utilizada a ferramenta DPCTL para recuperar os fluxos que possuem o nome de domínio casado com `www.dcc.ufmg.br`. Desta forma, é possível checar a quantidade de pacotes que casaram com a regra. Esta funcionalidade pode ajudar o administrador de rede à recuperar dados sobre os nomes de domínio acessados a partir de sua rede local. A quantidade de pacotes e outros dados do fluxo podem ser recuperados como demonstra o exemplo a seguir:

```
mininet> h1 wget www.dcc.ufmg.br

root@mininet-vm:~# dpctl dump-flows tcp:localhost:6634 dn_dst=www.dcc.ufmg.br
stats_reply (xid=0x1ed61a42): flags=none type=1(flow)
cookie=0,duration_sec=3s,duration_nsec=884000000,table_id=0,pirority=65535,
n_packets=35,n_bytes=2332,idle_timeout=10,hard_timeout=30,tcp,in_port=1,
dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=2e:b7:fd:84:f1:d9,
dl_dst=36:e6:64:bd:ce:2d,nw_src=10.0.0.1,nw_dst=150.164.0.135,
nw_tos=0x00,tp_src=40479,tp_dst=80,actions=output:3

root@mininet-vm:~# dpctl dump-flows tcp:localhost:6634 dn_src=www.dcc.ufmg.br
stats_reply (xid=0x15318be3): flags=none type=1(flow)
cookie=0,duration_sec=5s,duration_nsec=915000000,table_id=0,pirority=65535,
n_packets=62,n_bytes=80530,idle_timeout=10,hard_timeout=30,tcp,in_port=3,
dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=36:e6:64:bd:ce:2d,
dl_dst=2e:b7:fd:84:f1:d9,nw_src=150.164.0.135,nw_dst=10.0.0.1,
nw_tos=0x00,tp_src=80,tp_dst=40479,actions=output:1
```

6.9 Aplicação Prática: Bloqueando o WhatsApp

O serviço de mensagens WhatsApp possui 1 bilhão de usuários espalhados pelo mundo [8]. Este número de usuários foi alcançado no dia 1 de Fevereiro de 2016. Atu-

[8]https://blog.whatsapp.com/616/Um-bilhão?

almente, o serviço conta com 3867 endereços IPv4 [9] e 52 nomes de domínio utilizados pelo serviço [10]. Destes 52 nomes de domínio listados apenas 26 responderam requisição via `nslookup`. Foram recuperados 151 endereços IPs via `nslookup`. A tabela 6.11 apresenta os números relativos a endereços IPs e nomes de domínio do WhatsApp. Assim, foi criado um sistema para bloquear o WhatsApp. São comparadas duas formas de fazer este bloqueio. A primeira utilizando o DN+OpenFlow e a segunda tratando os nomes de domínio no controlador Pox utilizando a implementação padrão do OpenFlow.

Tabela 6.11. Quantidade de endereços IPs, Regras CIDR e Nomes de Domínio do WhatsApp

Endereços IPv4	Notação CIDR	Nomes de Domínio
3867	145	52

Para tratar os nomes de domínio no controlador foi criada uma aplicação baseada no módulo `DNS Spy` provido pelo Pox. O `DNS Spy` funciona da seguinte forma: todos pacotes DNS de resposta que chegam ao switch OpenFlow são encaminhados ao controlador. Os pacotes DNS são interpretados e são extraídos o nome de domínio e endereços IPs. Assim, é possível construir um Firewall para os endereços IPs extraídos utilizando seus nomes de domínio. Os nomes de domínio extraídos são comparados aos nomes em uma *blacklist*. Baseado nessa lista, é escolhida a ação para os endereços IPs. São adicionados à lista os 26 nomes de domínio que responderam requisição `nslookup`.

O módulo modificado funciona de forma similar ao DN+OpenFlow, o diferencial é que o processamento é realizado no controlador. Para realizar esta aplicação no DN+OpenFlow basta adicionar regras simples de bloqueio utilizando os 26 nomes de domínio. O apêndice B demonstra de forma técnica como realizar esta ação.

O Bloqueio utilizando apenas os endereços IPs (Firewall L3) listados também é possível, mas serão gastas 145 regras (regras CIDR) na tabela de Fluxos do switch OpenFlow. Este número estoura a quantidade máxima de regras ACL do switch testado. O switch OpenFlow Stanford Reference comporta apenas 100 regras não exatas. Isto é, regras que possuem campos sem preencher *(wildcards)*. Além disso, essa abordagem não resiste a mudanças nos endereços IPs utilizados, necessitando de manutenções periódicas nas regras instaladas. Por isso, iremos comparar apenas o DN+OpenFlow e a versão modificada do `DNS Spy` que foi intitulada pelos autores de `Firewall L7`.

A primeira comparação é a quantidade de regras utilizada. O DN+OpenFlow utiliza 26 regras apenas, cerca de 5 vezes menor a quantidade de regras necessárias pelo Firewall L3. Vale ressaltar que as regras do DN+OpenFlow são mais persistentes,

[9] https://www.whatsapp.com/cidr.txt
[10] https://github.com/ukanth/afwall/wiki/HOWTO-blocking-WhatsApp

pois, se for removido ou adicionado algum endereço IP as regras IPs terão que ser alterados. Diferentemente de nomes de domínio, que já são utilizados para agregar vários endereços IPs e dificilmente serão alterados. O administrador ganha em quantidade de regras no switch e persistência ao adotar a solução DN+OpenFlow. Já o Firewall L7 chega a estourar a tabela de fluxos do switch OpenFlow necessitando mais de 100 regras. A figura 6.16 apresenta a quantidade de regras necessárias para cada solução. O DN+OpenFlow utilizou uma quantidade de cerca 5 vezes menor de regras.

Figura 6.16. Quantidade de regras utilizadas para cada solução

Para avaliar a latência das aplicações foram alteradas as ações das regras para permitir o acesso aos servidores do WhatsApp. Foi utilizada a ferramenta `hping` [11]. Hping provê a capacidade de enviar sondas utilizando o protocolo TCP. Desta forma enviamos 33 sondas para cada nome de domínio, totalizando 858 sondas, foi utilizada a porta 443. O WhatsApp utiliza as portas 80, 443, 5222. A escolha da 443 foi devido à maior parte dos servidores responderem nessa porta. A figura 6.17 apresenta os resultados para a latência média obtida para cada solução.

A tabela 6.12 apresenta os valores plotados no gráfico 6.17. O DN+OpenFlow foi 48 milissegundos mais rápido que a solução Firewall L7, representando latência 24% menor. Isso se deve à necessidade de enviar cada pacote DNS de resposta para o controlador e só depois disso instalar as regras utilizando os endereços IPs no switch.

[11] http://www.hping.org/hping3.html

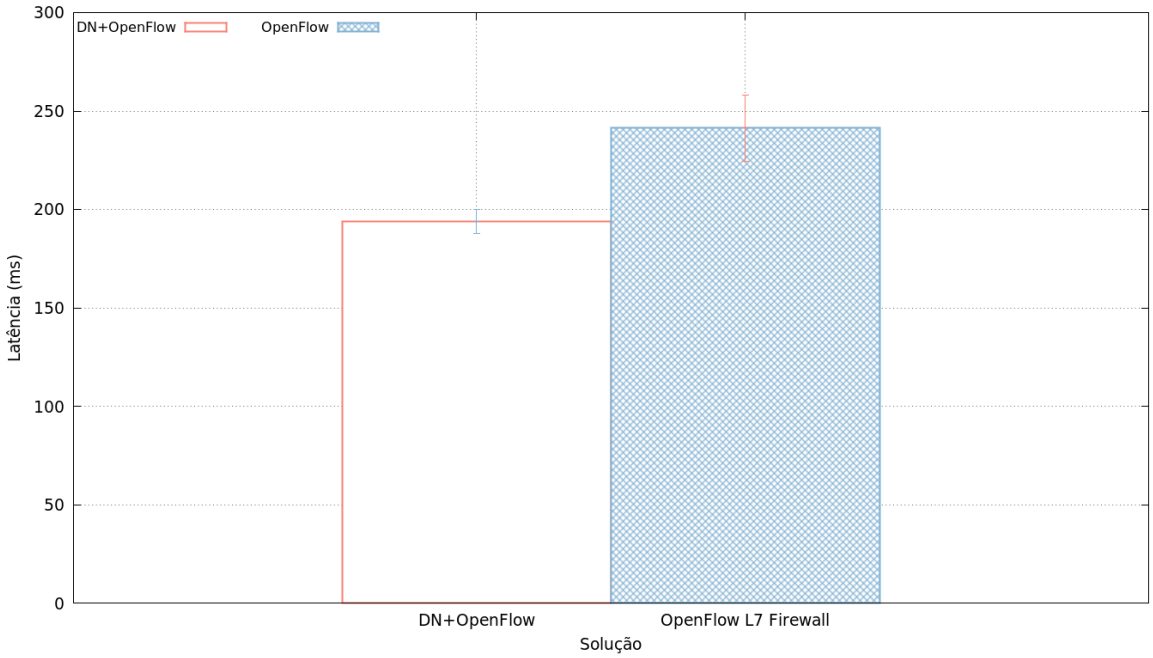


Figura 6.17. Latência média por aplicação

Além disso, a solução L7 Firewall gastou 26 pacotes a mais, um para cada nome de domínio consultado.

Tabela 6.12. Latência média por aplicação (ms)

Solução	Latência(ms)	Intervalo de confiança
DN+OpenFlow	193,95	± 6,03
OpenFlow L7 Firewall	241,40	± 16,92

A figura 6.18 apresenta os valores encontrados para a latência do experimento utilizando o Hping. É possível reparar que a linha representada pelo DN+OpenFlow possui valores menores para a vazão. Os valores para a solução OpenFlow L7 Firewall explodem à medida que a tabela do switch é totalmente preenchida. Isso explica os pontos com vazão próxima a 1200 ms.

A figura 6.19 apresenta um zoom em 70% dos pacotes de latência até 200ms. Desta forma é possível observar melhor o comportamento das soluções. DN+OpenFlow ganha em todos os pacotes quando se comparada a latência.

6.10 Resumo dos Resultados Obtidos

Nesta subseção apresentaremos o resumo dos resultados obtidos nesta dissertação. Os resultados foram divididos em três subseções. São elas: *Overhead*, onde serão tratados

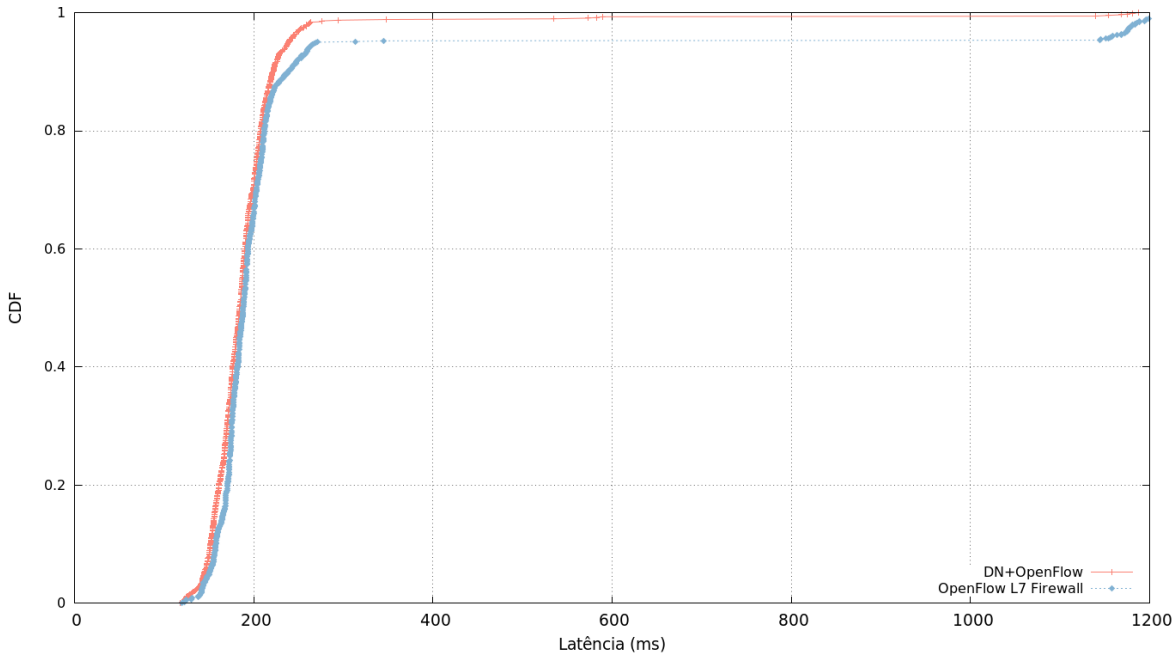


Figura 6.18. CDF Latência utilizando Hping

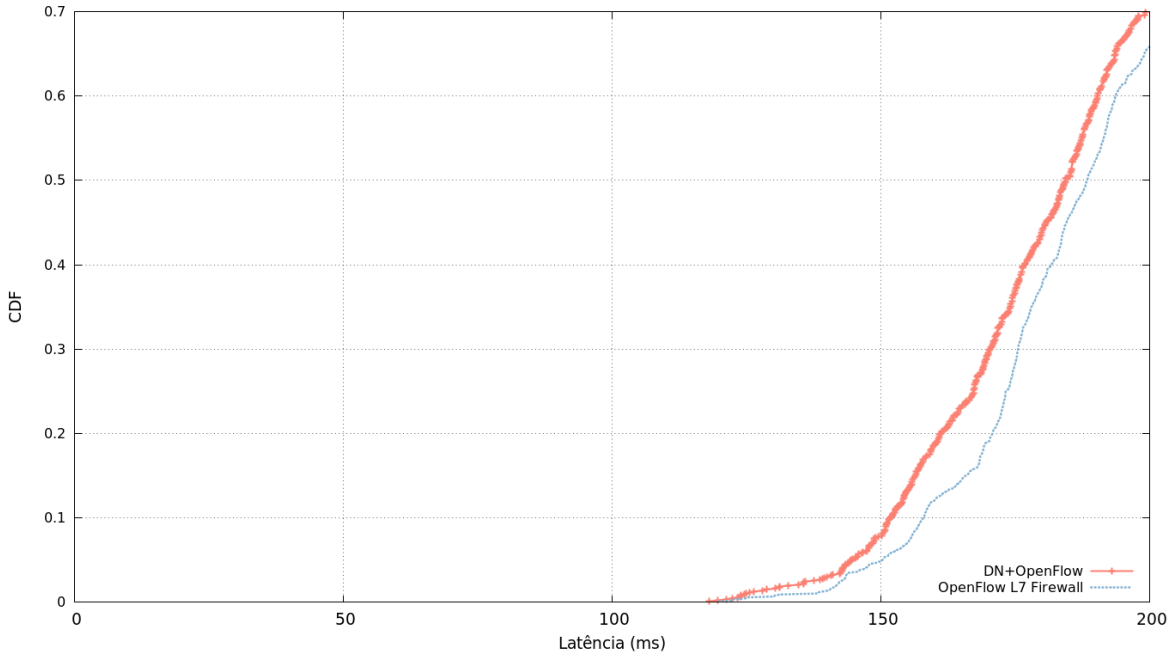


Figura 6.19. Zoom 70% dos pacotes - CDF Latência utilizando Hping

os custos adicionais que a solução DN+OpenFlow apresentou; Sem Alterações, são os resultados que não obtiveram alterações ao serem comparados com a solução padrão; E por último, os Benefícios que a solução DN+OpenFlow apresenta.

6.10.1 *Overhead*

A solução DN+OpenFlow obteve o custo adicional (*Overhead*) nos seguintes quesitos:

- 10% no consumo da CPU, contabilizando o agregado do Switch e Controlador.
- 13% na vazão nos testes utilizando iperf
- 19% no *page load time* testado em máquinas locais
- 8% no *page load time* testado na Internet
- Tabela <DN,IP> 352 bytes por linha e 5,28MB no total (15000 linhas)

6.10.2 Sem Alteração

O quesito que não apresentou alteração em comparação com o DN+OpenFlow e a solução padrão foi:

- Latência

6.10.3 Benefícios

A solução DN+OpenFlow apresenta benefícios nos seguintes quesitos nos casos especificados:

- Foi diminuída a quantidade de regras necessárias no switch para bloquear um domínio com vários endereços IPs
- Foi possível coletar dados estatísticos dos fluxos a partir do nome de domínio acessado
- Foi possível aumentar o grau de abstração dos programas SDN
- Foi possível diminuir a latência de aplicações que tratam nomes de domínio no controlador
- Foi possível diminuir a quantidade de pacotes enviados ao controlador (para aplicações que tratam nomes de domínio no controlador).

Capítulo 7

Conclusão e Trabalhos Futuros

A evolução do protocolo OpenFlow pode trazer várias oportunidades de inovação para redes de computadores, centralizando a administração de redes, melhorando a segurança e tornando-as mais programáveis. Nesta dissertação foi apresentada uma arquitetura que possibilita switches SDN casar fluxos utilizando nomes de domínio. A adição de regras de casamento baseadas em nomes de domínio é uma opção interessante, por exemplo, devido ao número de endereços IP associados apenas a um nome. Utilizar a arquitetura proposta para realizar regras de casamento é mais simples porque o *switch* vai lidar com todos endereços IP utilizando nomes de domínio.

As vantagens de nossa arquitetura são reduzir o número de pacotes enviados para o controlador, uma forma melhor para criar regras utilizando nomes de domínio, e fornecer um maior grau de abstração para programadores SDN. Além disso, nós podemos utilizar todas as funcionalidades do switch OpenFlow para coletar dados estatísticos utilizando nomes de domínio, criando relatórios baseados nos acessos realizados a *Web Sites* em uma rede específica e otimizar o seu uso.

Em uma aplicação apresentada nesta dissertação, onde o foco é bloquear o sistema de envio de mensagens Whatsapp. Foi possível comparar nossa solução com outras soluções e mostrar os benefícios da nossa arquitetura. Foi possível diminuir a latência em comparação com a solução comparada, diminuir a quantidade de regras na tabela do switch, e reduzir a quantidade de pacotes enviados ao controlador.

Outro ponto forte destacado neste projeto é como realizar a extensão do protocolo OpenFlow. Foi demonstrado como estender o switch e o controlador de uma arquitetura OpenFlow. Assim, qualquer projeto que pretenda alterar o protocolo OpenFlow pode se basear nesta dissertação como tutorial de como implementar a sua própria versão do OpenFlow.

Este trabalho também funciona como premissa para implementações baseadas em

redes orientadas a conteúdo [Jacobson et al., 2009]. Um roteador orientado a conteúdo pode ser implementado assim como o DN+OpenFlow foi. Os campos `dn_src/dn_dst` podem ser substituídos por campos que se referem a conteúdo de origem e destino. Por exemplo, pacotes orientados ao conteúdo `video` podem ser redirecionados para uma determinada porta do roteador, até o servidor correto.

Como trabalho futuro, nós pretendemos portar nossa implementação para um switch que trabalhe a nível de *kernel*, evitando a necessidade de copiar pacotes do *kernel* para espaço de usuário, melhorando o seu desempenho. Outro objetivo importante do nosso projeto é suportar mais controladores OpenFlow. Neste trabalho foi apresentado também um tutorial de como estender o protocolo OpenFlow para o controlador POX. Além disso, pretendemos avaliar a possibilidade de utilizar *wildcards* no campo nome de domínio. Nosso objetivo é levantar a viabilidade da implementação e o custo adicional que será acrescido ao switch.

Apêndice A

DN+OpenFlow

Neste apêndice estão os trechos de código mais importantes para esta dissertação. Vale ressaltar que todo o código está disponível no repositório do GitHub [1]. Para instalar o switch DN+OpenFlow basta seguir as seguintes instruções:

```
# git clone github.com/lucasmaiasilva/dn-openflow
# cd dn-openflow
# ./boot
# ./configure
# make
# make install
```

Para executar o DN+OpenFlow utilizando o Mininet basta seguir os seguintes passo:

```
# mn --switch user --controller remote --nat --listenport 6634
```

A.1 Estrutura de Casamento DN+OpenFlow

A nova estrutura de casamento do DN+OpenFlow pode ser conferida no trecho de código abaixo:

Código A.1. Estrutura ofp_match

```
/* Fields to match against flows */
#define DN_SIZE 256
struct ofp_match {
```

[1] github.com/lucasmaiasilva/dn-openflow

```
  uint32_t wildcards;
  uint16_t in_port;
  uint8_t  dl_src[OFP_ETH_ALEN];
  uint8_t  dl_dst[OFP_ETH_ALEN];
  uint16_t dl_vlan;
  uint8_t  dl_vlan_pcp;
  uint8_t  pad1[1];
  uint16_t dl_type;
  uint8_t  nw_tos;
  uint8_t  nw_proto;
  uint8_t  pad2[2];
  uint32_t nw_src;
  uint32_t nw_dst;
  uint16_t tp_src;
  uint16_t tp_dst;
  uint8_t  dn_src[DN_SIZE];
  uint8_t  dn_dst[DN_SIZE];
};
OFP_ASSERT(sizeof(struct ofp_match) == 552);
```

Código A.1. Estrutura ofp_match

A.2 O Parser DNS do DN+OpenFlow

Código A.2. Funções Parser DNS

```
void parse_dns_name(struct ofpbuf *b, uint8_t name[]){

  uint8_t *pull=0x1;
  memset(name,'\0',256);
  pull = ofpbuf_try_pull(b,1);
  if(pull==NULL){
    return;
  }
  /*trata labels*/
  if(pull[0]==0xc0){
```

```
    pull = ofpbuf_try_pull(b,1);
    if(pull==NULL){
      return;
    }
  }
  else{
    while(pull[0]!=0x0){
      pull=ofpbuf_try_pull(b,1);
      if(pull==NULL){
        return;
      }
      if(pull[0]<0x20&&pull[0]>0x0){
        name[strlen(name)]=0x2e;
      }else{
        memcpy(name+strlen(name),pull,1);
      }
    }
  }
}

void dns_parser(struct ofpbuf b,struct tabela tab[]){
  uint8_t name[512];
  uint8_t aux[1024];
  uint32_t *auxiliar;
  int i=0;
  char *ipv4;
  struct in_addr *ptr;

  /*retira cabecalho do pacote dns 12 bytes fixos do
     cabecalho*/
  const struct dns_header *dns = ofpbuf_try_pull(&b,12);
  /*trata erro do servidor*/
  if((ntohs(dns->flags)&0x2)==0x2){
    return;
  }
```

```
/*trata o nome da consulta dns*/
parse_dns_name(&b,&name);
/*retira os dados da questao DNS*/
const struct dns_question *dns_q = ofpbuf_try_pull(&b,4);
for(i=0;i<ntohs(dns->n_answers);i++){
  parse_dns_name(&b,&aux);
  const struct dns_ans_header *dns_a =
     ofpbuf_try_pull(&b,10);
  if(dns_a==NULL){
    return;
  }
  auxiliar = ofpbuf_try_pull(&b,ntohs(dns_a->data_len));
  if (auxiliar==NULL){
    return;
  }
  if((ntohs(dns_a->type)==1)&&(ntohs(dns_a->class)==1)){
    ptr=auxiliar;
    ipv4 = inet_ntoa(*ptr);
    if(busca(tab,*auxiliar)==0){
      adiciona(tab,name,*auxiliar,ntohl(dns_a->ttl));
    }
  }
}

/*authority ns*/
for(i=0;i<ntohs(dns->n_aut_rec);i++){
  parse_dns_name(&b,&aux);
  const struct dns_ans_header *dns_a =
     ofpbuf_try_pull(&b,10);
  if(dns_a==NULL){
    return;
  }
  auxiliar = ofpbuf_try_pull(&b,ntohs(dns_a->data_len));
  if (auxiliar==NULL){
    return;
  }
```

```
    if((ntohs(dns_a->type)==1)&&(ntohs(dns_a->class)==1)){
      ptr=auxiliar;
      ipv4 = inet_ntoa(*ptr);
      if(busca(tab,*auxiliar)==0){
        adiciona(tab,name,*auxiliar,ntohl(dns_a->ttl));
      }
    }
  }

  /*additional records*/

  for(i=0;i<ntohs(dns->n_rec_pkt);i++){
    parse_dns_name(&b,&aux);
    const struct dns_ans_header *dns_a  =
       ofpbuf_try_pull(&b,10);
    if(dns_a==NULL){
      return;
    }
    auxiliar = ofpbuf_try_pull(&b,ntohs(dns_a->data_len));
    if (auxiliar==NULL){
      return;
    }
    if((ntohs(dns_a->type)==1)&&(ntohs(dns_a->class)==1)){
      ptr=auxiliar;
      ipv4 = inet_ntoa(*ptr);
      if(busca(tab,*auxiliar)==0){
        adiciona(tab,name,*auxiliar,ntohl(dns_a->ttl));
      }
    }
  }
  //print_dns(dns,name,dns_q,dns_a1);
  //imprimeTabela(tab);
}
```

Código A.2. Funções Parser DNS

A.3 Tabela <IP, Nome de Domínio>

Código A.3. Estrutura e Funções Tabela IP Nome de Domínio

```
struct tabela{
  uint8_t dn[256];
  uint32_t ip;
  uint32_t ttl;
  uint32_t ttl_old;
};


void apaga(struct tabela tab[]){
  int i;
  for(i=0;i<tamanho;i++){
    if(tab[i].ttl==0){
      tab[i].ip = tab[tamanho-1].ip;
      memcpy(tab[i].dn, tab[tamanho-1].dn,512);
      tab[i].ttl = tab[tamanho-1].ttl;
      tamanho--;
    }
  }
}

void adiciona(struct tabela tab[],uint8_t dn[512],
   uint32_t ip,uint32_t ttl){
  memcpy(tab[tamanho].dn,dn,256);
  tab[tamanho].ip=ip;
  tab[tamanho].ttl=ttl;
  tab[tamanho].ttl_old=ttl;
  tamanho++;
}

void imprimeTabela(struct tabela tab[]){
  FILE *arquivo;
  char *ipv4;
  struct in_addr *ptr;
  int i = 0;
  arquivo=fopen("tabela","w+");
```

```
  for(i=0;i<tamanho;i++){
    ptr=&tab[i].ip;
    fprintf(arquivo,"dn - %s ip - %x ipv4 - %s ttl -
       %d\n",tab[i].dn,tab[i].ip,inet_ntoa(*ptr),tab[i].ttl);
  }
  fprintf(arquivo,"tamanho %d\n",tamanho);
  fclose(arquivo);

}

int tabelaCmpIp(const void *v1, const void *v2){
  const struct tabela *t1 = v1;
  const struct tabela *t2 = v2;

  if (t1->ip == t2->ip)
    return 0;
  if(t1->ip > t2->ip)
    return 1;
  if(t1->ip < t2->ip)
    return -1;
}

uint8_t* buscaIp(struct tabela tab[],uint32_t ip){
  struct tabela item, *resultado;
  item.ip=ip;
  qsort(tab,tamanho,sizeof(struct tabela),tabelaCmpIp);
  resultado = bsearch (&item, tab, tamanho, sizeof (struct
     tabela),tabelaCmpIp);
  if (resultado){
    resultado->ttl=resultado->ttl_old;
        return resultado->dn;
  }
  else{
    return NULL;
  }
}
```

```
int busca(struct tabela tab[],uint32_t ip){
  struct tabela item, *resultado;
  item.ip=ip;
  qsort(tab,tamanho,sizeof(struct tabela),tabelaCmpIp);
  resultado = bsearch (&item, tab, tamanho, sizeof (struct
     tabela),tabelaCmpIp);
  if (resultado)
    return 1;
  else
    return 0;
}
```

Código A.3. Estrutura e Funções Tabela IP Nome de Domínio

Apêndice B

Código DN+POX

Neste apêndice são dados exemplos de como utilizar o DN+Pox. O código está disponível no repositório do GitHub [1]. Para utilizá-lo basta seguir os seguints passos:

```
# git clone github.com/lucasmaiasilva/dn-pox
# cd dn-pox
# ./pox.py forwarding.l2_learning
```

Para executar o DN+Pox utilizando o Mininet basta seguir os seguintes passo:

```
# mn --switch user --controller remote --nat --listenport 6634
```

B.1 Exemplo de utilização do DN+POX

Um exemplo de como pode ser utilizado o DN+POX é a criação de um Firewall. O Firewall tem como objetivo bloquear todos os pacotes de um dado endereço. Um Firewall L3 seria bloqueio para todos os pacotes de um endereço IP. O Firewall criado pelo DN+POX trabalha como um Firewall L3, porém com uma abrangência maior para sites que possuem mais de um endereço IP e com uma quantidade menor de regras.

Código B.1. Exemplo de uso DN+POX

```
1 def _handle_ConnectionUp (self, event):
2   if self._install_flow:
3     msg = of.ofp_flow_mod()
4     msg.match.dn_dst="www.uol.com.br"
```

[1] github.com/lucasmaiasilva/dn-pox

```
5       msg.actions.append(of.ofp_action_output(port =
            of.OFPP_NONE))
6       event.connection.send(msg)
```

A linha **1** do código exibido em B.2 demonstra o método que irá tratar o início de uma conexão. A linha **2** trata a instalação de um novo fluxo assim que a conexão é iniciada. A linha **3** cria o objeto `msg` que é do tipo `OFP_FLOW_MOD`. Esse objeto representará uma nova entrada de fluxo na tabela do switch. A linha **4** atribui o nome de domínio de destino para o campo de casamento do novo fluxo. A linha **5** apresenta a ação associada àquele fluxo. Neste caso, a ação é de descarte de pacote, a constante `OFPP_NONE` possui o valor `65535` que corresponde um número de porta utilizado para descarte de pacote. Neste caso, a linha de número **5** não é obrigatória, a ausência de ação faz com que os pacotes que casem com aquele fluxo sejam descartados. Por fim, a linha de número **6** envia a nova regra de fluxo para o switch.

O exemplo B.2 demonstra como criar um Firewall proativo. Este tipo de Firewall é chamado de proativo de acordo com a forma de como as regras são inseridas no switch. As regras são inseridas na inicialização da conexão do switch com o controlador, independente de qualquer evento que aconteça na rede. Outro modo de escrever um Firewall seria com a abordagem reativa. Esta abordagem trata os pacotes de acordo com os eventos da rede. O código completo do Firewall exibido nesta seção pode ser conferido no apêndice da dissertação e no repositório deste projeto no Github [2].

Código B.2. Exemplo de uso DN+POX

```
1  from pox.core import core
2  import pox.openflow.libopenflow_01 as of
3  import pox.lib.packet as pkt
4  import pox.lib.packet.dns as pkt_dns
5  from pox.lib.addresses import IPAddr
6  from pox.lib.revent import *
7
8  class dn_openflow(EventMixin):
9    def __init__(self, install_flow = True):
10     #Event.__init__()
11     self._install_flow = install_flow
12     core.openflow.addListeners(self)
13
14
```

[2] https://github.com/lucasmaiasilva/dn-pox

```
15    def _handle_ConnectionUp (self, event):
16      if self._install_flow:
17        msg = of.ofp_flow_mod()
18        msg.match = of.ofp_match()
19        msg.match.dn_dst="web.whatsapp.com"
20        msg.idle_timeout = 65535
21        msg.priority = 65535
22        msg.actions.append(of.ofp_action_output(port =
              of.OFPP_NONE))
23        event.connection.send(msg)
24        print "regra de destino inserida ", msg
25
26    def _handle_PacketIn (self, event):
27      print "entrada de pacote"
28
29  def launch(no_flow = False):
30    core.registerNew(dn_openflow,not no_flow)
```

Referências Bibliográficas

Andreolini, M.; Colajanni, M. & Nuccio, M. (2003). Kernel-based Web Switches Providing Content-aware Routing. Em *Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on*, pp. 25–32.

Apostolopoulos, G.; Aubespin, D.; Peris, V.; Pradham, P. & Saha, D. (2000). Design, Implementation and Performance of a Content-based Switch. Em *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pp. 1117–1126 vol.3. ISSN 0743-166X.

Bavier, A.; Bowman, M.; Chun, B.; Culler, D.; Karlin, S.; Muir, S.; Peterson, L.; Roscoe, T.; Spalink, T. & Wawrzoniak, M. (2004). Operating system support for planetary-scale network services. Em *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pp. 19–19, Berkeley, CA, USA. USENIX Association.

Berde, P.; Gerola, M.; Hart, J.; Higuchi, Y.; Kobayashi, M.; Koide, T.; Lantz, B.; O'Connor, B.; Radoslavov, P.; Snow, W. & Parulkar, G. (2014). ONOS: Towards an Open, Distributed SDN OS. Em *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pp. 1–6, New York, NY, USA. ACM.

Bernaschi, M.; Casadei, F. & Tassotti, P. (2007). Sockmi: A Solution for Migrating TCP/IP Connections. Em *Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on*, pp. 221–228. ISSN 1066-6192.

Bezahaf, M.; Alim, A. & Mathy, L. (2013). FlowOS: A Flow-based Platform for Middleboxes. Em *Proceedings of the 2013 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMiddlebox '13, pp. 19–24, New York, NY, USA. ACM.

Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G. & Walker, D. (2014). P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95. ISSN 0146-4833.

Bremler-Barr, A.; Harchol, Y. & Hay, D. (2015). Openbox: Enabling Innovation in Middlebox Applications. Em *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMiddlebox '15, pp. 67–72, New York, NY, USA. ACM.

Bremler-Barr, A.; Harchol, Y. & Hay, D. (2016). Openbox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions. Em *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM '16, pp. 511–524, New York, NY, USA. ACM.

Bremler-Barr, A.; Harchol, Y.; Hay, D. & Koral, Y. (2014). Deep Packet Inspection as a Service. Em *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 271–282. ACM.

Casado, M.; Freedman, M. J.; Pettit, J.; Luo, J.; McKeown, N. & Shenker, S. (2007). Ethane: Taking Control of the Enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12. ISSN 0146-4833.

Chiosi, M.; Clarke, D.; Willis, P.; Reid, A.; Feger, J.; Bugenhagen, M.; Khan, W.; Fargano, M.; Cui, D. C.; Deng, D. H.; Benitez, J.; Michel, U.; Damker, H.; Ogaki, K.; Matsuzaki, T.; Fukui, M.; Shimano, K.; Delisle, D.; Loudier, Q.; Kolias, C.; Guardini, I.; Demaria, E.; Minerva, R.; Manzalini, A.; López, D.; Salguero, F. J. R.; Ruhl, F. & Sen, P. (2012). Network Functions Virtualisation an Introduction, Benefits, Enablers, Challenges & Call for Action. *SDN and OpenFlow World Congress.*

Collings, J. & Liu, J. (2014). An Openflow-Based Prototype of SDN-Oriented Stateful Hardware Firewalls. Em *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pp. 525–528.

Davie, B. S. & Farrel, A. (2008). *MPLS: Next Steps.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 9780080558295.

Feamster, N.; Rexford, J. & Zegura, E. (2013). The Road to SDN. *Queue*, 11(12):20:20–20:40. ISSN 1542-7730.

Gill, P.; Jain, N. & Nagappan, N. (2011). Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. *SIGCOMM Comput. Commun. Rev.*, 41(4):350–361. ISSN 0146-4833.

Goransson, P. & Black, C. (2014). *Software Defined Networks: A Comprehensive Approach*. Morgan Kaufmann. ISBN 978-0124166752.

Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N. & Shenker, S. (2008). NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110. ISSN 0146-4833.

Handigol, N.; Heller, B.; Jeyakumar, V.; Lantz, B. & McKeown, N. (2012). Reproducible Network Experiments Using Container-based Emulation. Em *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pp. 253–264, New York, NY, USA. ACM.

Intel (2014). Service-aware Network Architecture Based on SDN, NFV, and Network Intelligence.

Jacobson, V.; Smetters, D. K.; Thornton, J. D.; Plass, M. F.; Briggs, N. H. & Braynard, R. L. (2009). Networking Named Content. Em *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pp. 1–12, New York, NY, USA. ACM.

Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; Zolla, J.; Hölzle, U.; Stuart, S. & Vahdat, A. (2013). B4: Experience with a Globally-deployed Software Defined WAN. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14. ISSN 0146-4833.

Kachris, C. & Vassiliadis, S. (2006). Design of a Web Switch in a Reconfigurable Platform. Em *Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, ANCS '06, pp. 31–40, New York, NY, USA. ACM.

Koponen, T.; Casado, M.; Gude, N.; Stribling, J.; Poutievski, L.; Zhu, M.; Ramanathan, R.; Iwata, Y.; Inoue, H.; Hama, T. & Shenker, S. (2010). Onix: A Distributed Control Platform for Large-scale Production Networks. Em *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pp. 351–364, Berkeley, CA, USA. USENIX Association.

Kreutz, D.; Ramos, F.; Esteves Verissimo, P.; Esteve Rothenberg, C.; Azodolmolky, S. & Uhlig, S. (2015). Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76. ISSN 0018-9219.

Lantz, B.; Heller, B. & McKeown, N. (2010). A Network in a Laptop: Rapid Prototyping for Software-defined Networks. Em *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pp. 19:1–19:6, New York, NY, USA. ACM.

Li, H.; Hu, C.; Hong, J.; Chen, X. & Jiang, Y. (2015). Parsing Application Layer Protocol with Commodity Hardware for SDN. Em *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '15, pp. 51–61, Washington, DC, USA. IEEE Computer Society.

Liu, C. & Albitz, P. (1996). *DNS and BIND (1st Edition)*. O'Reilly Media, Inc. ISBN 1565920104.

McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S. & Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74. ISSN 0146-4833.

Mockapetris, P. (1987). *RFC 1035 Domain Names - Implementation and Specification*. Internet Engineering Task Force.

Mogul, J. C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Curtis, A. R. & Banerjee, S. (2010). DevoFlow: cost-effective flow management for high performance enterprise networks. Em *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pp. 1:1–1:6, New York, NY, USA. ACM.

Naous, J.; Erickson, D.; Covington, G. A.; Appenzeller, G. & McKeown, N. (2008). Implementing an Openflow Switch on the NetFPGA Platform. Em *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pp. 1–9, New York, NY, USA. ACM.

ONF (2014). Software-Defined Networking (SDN) Definition - What is SDN? `https://www.opennetworking.org/sdn-resources/sdn-definition`.

Open Networking Foundation (2012). Software-Defined Networking: The New Norm for Networks. White paper, Open Networking Foundation, Palo Alto, CA, USA.

Qazi, Z. A.; Tu, C.-C.; Chiang, L.; Miao, R.; Sekar, V. & Yu, M. (2013). Simplefying Middlebox Policy Enforcement Using SDN. Em *ACM SIGCOMM Computer Communication Review*, volume 43, pp. 27–38. ACM.

Ram, K. K.; Mudigonda, J.; Cox, A. L.; Rixner, S.; Ranganathan, P. & Santos, J. R. (2010). sNICh: Efficient Last Hop Networking in the Data Center. Em *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications*, ANCS '10, pp. 26:1–26:12. ACM.

Sekar, V.; Egi, N.; Ratnasamy, S.; Reiter, M. K. & Shi, G. (2012). Design and Implementation of a Consolidated Middlebox Architecture. Em *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pp. 323–336, San Jose, CA. USENIX.

Sekar, V.; Ratnasamy, S.; Reiter, M. K.; Egi, N. & Shi, G. (2011). The Middlebox Manifesto: Enabling Innovation in Middlebox Deployment. Em *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pp. 21:1–21:6, New York, NY, USA. ACM.

Sherry, J.; Hasan, S.; Scott, C.; Krishnamurthy, A.; Ratnasamy, S. & Sekar, V. (2012). Making Middleboxes Someone else's Problem: Network Processing as a Cloud Service. *SIGCOMM Comput. Commun. Rev.*, 42(4):13–24. ISSN 0146-4833.

Song, H. (2013). Protocol-oblivious Forwarding: Unleash the Power of SDN Through a Future-proof Forwarding plane. Em *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 127–132. ACM.

Suh, M.; Park, S. H.; Lee, B. & Yang, S. (2014). Building Firewall Over the Software-Defined Network Controller. Em *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, pp. 744–748.

Udechukwu, R. & Dutta, R. (2014). Extending Openflow for Service Insertion and Payload Inspection. Em *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pp. 589–595. IEEE.

White, B.; Lepreau, J.; Stoller, L.; Ricci, R.; Guruprasad, S.; Newbold, M.; Hibler, M.; Barb, C. & Joglekar, A. (2002). An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270. ISSN 0163-5980.

Yiakoumis, Y.; Katti, S. & McKeown, N. (2016). Neutral Net Neutrality. Em *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference*, SIGCOMM '16, pp. 483–496, New York, NY, USA. ACM.