UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
ENGENHARIA DE COMPUTAÇÃO

GUSTAVO MIOTTO

AppFlow: Suporte a Regras da Camada de Aplicação em Arquiteturas SDN OpenFlow

Trabalho de Graduação apresentado como
requisito parcial para a obtenção do grau de
Engenheiro de Computação

Prof. Dr. Marinho Pilla Barcellos
Orientador

Porto Alegre, Dezembro de 2014

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Miotto, Gustavo

AppFlow: Suporte a Regras da Camada de Aplicação em Arquiteturas SDN OpenFlow / Gustavo Miotto. – Porto Alegre: UFRGS, 2014.

41 f.: il.

Trabalho de Conclusão – Universidade Federal do Rio Grande do Sul. Engenharia de Computação, Porto Alegre, BR–RS, 2014. Orientador: Marinho Pilla Barcellos.

1. SDN. 2. OpenFlow. 3. Camada de aplicação. I. Barcellos, Marinho Pilla.

“There is no perfection, only life.”
— MILAN KUNDERA

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
DPI	*Deep Packet Inspection*
HTTP	*Hypertext Transfer Protocol*
NAT	*Network Address Translator*
NOS	*Network Operating System*
OSI	*Open Systems Interconnection*
RTT	*Round-trip time*
SDN	*Software-Defined Networking*
TCP	*Transport Control Protocol*
TLV	*Type-Length-Value*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Software-Defined Networking oferece a oportunidade para solucionar diversos problemas relacionados ao gerenciamento e inovação das redes de computadores. Para isso, um dos pilares de SDN é a programabilidade remota dos dispositivos de rede em nível de fluxos. A adoção de SDN vem sendo impulsionada pelo protocolo OpenFlow, o qual padroniza a comunicação remota com os dispositivos e define como deve ser feita a programação de fluxos dos mesmos.

Uma grande limitação presente atualmente em SDN se deve ao fato de que a programabilidade está restrita aos cabeçalhos dos níveis 2 (Enlace), 3 (Rede) e 4 (Transporte). Essa característica de projeto considera apenas roteadores e comutadores simples, ignorando os *middleboxes*. Dessa forma, as arquiteturas atuais não oferecem benefícios para componentes mais complexos, tais como *firewalls* e DPIs. Este trabalho propõe AppFlow, um projeto de extensão da arquitetura SDN com suporte a regras em nível de aplicação e um passo em direção à integração de funcionalidades de *middleboxes* aos dispositivos de encaminhamento. A viabilidade de AppFlow é verificada por meio de um protótipo com suporte ao protocolo HTTP. Experimentos são realizados com o simulador GNS3.

Palavras-chave: SDN, OpenFlow, Camada de aplicação.

AppFlow: Support for Application Layer Rules in SDN OpenFlow Architectures

ABSTRACT

Software-Defined Networking offers the opportunity to solve many problems related to the management and innovation of computer networks. For this, one of the pillars of SDN is remote programmability of network devices in flow level. The adoption of SDN has been driven by the OpenFlow protocol, which standardizes the remote communication with devices and defines how the programming of his flows should be done.

A major limitation currently present in SDN is due to the fact that the programmability is restricted to the headings of level 2 (Link), 3 (Network) and 4 (Transport). This design feature considers only routers and simple switches, ignoring the middleboxes. Thus, current architectures do not provide benefits to more complex components such as firewalls and DPIs. This work proposes AppFlow, an extension project of the SDN architecture with support for application-level rules and a step towards the integration of middleboxes features to forwarding devices. The AppFlow viability is checked by means of a prototype to support the HTTP protocol. Experiments are performed with the GNS3 simulator.

Keywords: SDN, OpenFlow, Application Layer.

1 INTRODUÇÃO

O paradigma Redes Definidas por Software (*Software-Defined Networking* - SDN) busca superar as dificuldades em evoluir e administrar as redes de computadores de atuais. Para tanto, o paradigma é fundamentado no conceito de separação entre os planos de controle e de dados. Essa separação ocorre da seguinte forma: enquanto o plano de dados permanece nos dispositivos de rede para realizar as funções de encaminhamento, o plano de controle é removido dos mesmos e centralizado. Como resultado, os dispositivos de rede são simplificados (uma vez que não precisam conter algoritmos complexos do plano de controle) e a complexidade da infraestrutura, atribuída principalmente às funções do plano de controle, passa a ser implementada em software e sobre uma lógica centralizada (NUNES et al., 2014). Pragmaticamente, essa separação é dada por uma interface padrão de configuração remota, exposta por meio de uma API (Application Programming Inteface), que permite ao plano de controle definir as regras de encaminhamento utilizadas pelo plano de dados. A adoção do paradigma é possibilitada pelo padrão OpenFlow, o qual define a API e quais tipos de campos e ações podem ser definidos nas tabelas dos dispositivos (MCKEOWN et al., 2008).

Em geral, redes SDN são implementadas com base em uma arquitetura de três camadas: dispositivos de encaminhamento, controlador e serviços de controle (LARA; KOLASANI; RAMAMURTHY, 2014). Os dispositivos de encaminhamento são responsáveis apenas por comutar os pacotes e se comunicar com o controlador; diferentemente dos dispositivos tradicionais, eles não executam as funções de controle [1] internamente. O controlador recebe informações de estado dos dispositivos físicos e provê uma camada de abstração para a programação remota destes dispositivos. Esse elemento pode exercer controle direto sobre o estado do plano de dados através da definição de regras na tabela de encaminhamento dos dispositivos. Por sua vez, os serviços de controle executam os algoritmos que irão definir o comportamento da rede sobre as abstrações providas pelo controlador.

Apesar de serem altamente flexíveis, as arquiteturas SDN estão limitadas a regras que operem sobre os cabeçalhos nos níveis 2 (Enlance), 3 (Rede) e 4 (Transporte) (ONF, 2013). Portanto, na prática, esse modelo se restringe a redes formadas apenas por roteadores e comutadores simples. Redes de computadores comumente possuem diversos dispositivos mais completos, denominados *middleboxes*, que realizam funções complexas na rede, por exemplo, *Deep Packet Inspection*, *firewall*, *Network Address Translators* (NATs), etc. Tais *middleboxes* executam em computadores ou dispositivos especializados, e estão localizados juntamente com roteadores ou *switches* (QAZI et al., 2013). Em linha com a filosofia de SDN, que busca simplificar a infraestrutura das redes, regras que

[1] Ex.: algoritmos de roteamento, balanceamento de carga e controle de acesso.

operam sobre o nível de aplicação (5+) são de grande utilidade para implementação desses serviços complexos. A existência das mesmas pode eliminar a necessidade de alguns *middleboxes* e/ou possibilitar o desenvolvimento de *middleboxes* mais adequados às necessidades específicas de cada rede em particular (ao invés do uso de *middleboxes* com funcionalidades genéricas).

Este trabalho propõe um estudo sobre a viabilidade da programação de tabelas em nível de aplicação (5+) para redes SDN OpenFlow. Esse estudo provê três principais contribuições:

- **AppFlow, um projeto de extensão da arquitetura SDN que comporta especificação de regras e ações em nível de aplicação**. AppFlow usa como base o protocolo OpenFlow e controladores e dispositivos que utilizam esse protocolo. Essa escolha tem em vista manter compatibilidade com a maior parte das arquiteturas atuais, visto que OpenFlow é o padrão mais adotado para a programação remota de dispositivos em redes SDN.

- **Um protótipo implementando AppFlow com protocolo HTTP**. O protótipo permite a validação da arquitetura com um protocolo amplamente utilizado e estudado. Apesar do protótipo ser orientado ao protocolo HTTP, a arquitetura AppFlow é genérica o suficiente para implementar qualquer protocolo que possua campos seguindo uma estrutura <Chave, Valor> bem definida (por exemplo, DNS).

- **Experimentação.** São realizados experimentos em ambiente emulado com uso do Simulador GNS3. Os resultados permitem a análise em ambiente controlado e visam demonstrar a factibilidade do protótipo.

Além dessas contribuições são discutidas algumas tomadas de decisões relacionadas a ideia inicial (apresentada no TG1). Conforme será discutido, a complexidade do problema guiou a escolha por uma abordagem mais genérica, em termos arquiteturais, e mais específica do ponto de vista da implementação. Ademais, as mudanças também afetaram o software utilizado como base para a implementação do trabalho.

O restante do trabalho está organizado como segue. O Capítulo 2 apresenta o estado da arte em SDN, explicando seu funcionamento e arquitetura, e descreve motivação, desafios e propostas relacionadas a regras em nível de aplicação. O Capítulo 3 descreve o projeto do AppFlow, incluindo as dificuldades encontradas e as soluções adotadas. No Capítulo 4, são discutidos a metodologia de avaliação, o ambiente utilizado e os resultados obtidos. Por fim, o Capítulo 5 conclui o documento com as principais conclusões e planos de trabalhos futuros.

2 ESTADO DA ARTE

Este capítulo introduz o estado-da-arte em SDN e esforços que visam a incorporar regras em nível de aplicação ao paradigma. Para isso, primeiramente a Seção 2.1 apresenta uma visão geral de arquiteturas SDN recentes. A seguir, a Seção 2.2 expõe conceitos relacionados a regras em nível de aplicação e os trabalhos relacionados.

2.1 Software-Defined Networking

A ideia principal por trás de Redes Definidas por Software é a desassociação entre o plano de dados e o plano de controle. Essa separação tem por objetivo simplificar a administração da rede e, assim, permitir redes mais flexíveis e customizáveis (JARRAYA; MADI; DEBBABI, 2014). Tal mudança surgiu pela dificuldade em evoluir a rede atual, que se diz estar "ossificada" (MCKEOWN et al., 2008), devido a forte integração entre o plano de dados e o de controle.

Este trabalho segue a definição de SDN em quatro pilares principais, conforme proposto na literatura atual (KREUTZ et al., 2014):

1. O plano de dados e de controle estão desassociados.

2. A lógica de controle é movida para uma entidade externa, denominada de controlador SDN.

3. Decisões de encaminhamento são baseadas em fluxos definidos por tuplas nos cabeçalhos dos pacotes, ao invés de utilizar somente o endereço destino.

4. A rede é programável através de serviços (aplicações) que executam sobre o controlador.

De maneira simplificada, o paradigma pode ser descrito da seguinte forma. Os dispositivos de encaminhamento passam a ser simples comutadores de pacotes e toda a decisão de encaminhamento é feita através de um controlador central. Esse controlador tem visão completa da topologia da rede e gera regras de encaminhamentos baseadas na abstração de fluxos. Um fluxo é identificado por tuplas compostas por informações de cabeçalhos comuns das camadas L2-L4 (ex.: tags VLAN e/ou MPLS, MAC origem/destino, IP origem/destino, porta TCP/UDP, etc). Dessa forma, os dispositivos de encaminhamento realizam comutação de pacotes baseados nas regras definidas em sua tabela de fluxos, onde tem-se uma regra e uma ação correspondente (ex.: porta de saída <número>, descartar, trocar endereço IP destino, etc). Caso algum pacote não "case" com qualquer regra, o mesmo é enviado ao controlador, o qual, por sua vez, decidirá o que deve ser realizado.

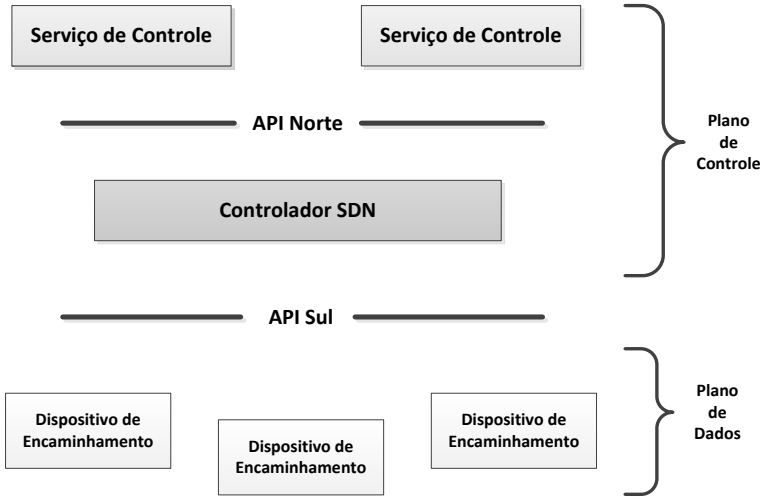

Figura 2.1: Arquitetura SDN.

Arquiteturas SDN podem ser representadas por cinco elementos principais (ONF, 2014a), conforme observado na Figura 2.1. A seguir descreve-se cada componente.

Serviços de Controle. Os Serviços de Controle são responsáveis por determinar o comportamento da rede, definindo, para isso, políticas de fluxos. Tais especificações são feitas em alto nível e passadas para o controlador. Dentre as vantagens desse tipo de abstração é a simplificação na composição e interação entre aplicações. Ademais, as aplicações fazem uso de uma API que provê visão global da rede como abstração, tornando-se mais fácil a criação de políticas complexas.

API Norte. API responsável por fornecer os recursos disponíveis pelo controlador para a aplicação. Essa API não é bem definida e cada controlador pode definir uma específica.

Controlador. Também chamado de Network Operating System (NOS), o controlador é responsável por fornecer abstrações para a coleta de estado da rede, visão global e facilitar a programação dos dispositivos de encaminhamento através de uma lógica centralizada. Exemplos: NOX (NOXRepo.org , 2014a), POX (NOXRepo.org , 2014b), Floodlight (Big Switch Networks, 2014), Ryu (Ryu, 2014).

API Sul. API responsável pela comunicação entre os dispositivos de encaminhamento e o controlador. A API mais amplamente adotada é a definida no protocolo OpenFlow (ONF, 2014b).

Dispositivos de encaminhamento. Consiste no plano de dados da rede. Os dispositivos de encaminhamento são responsáveis por comutar os pacotes na rede e manter a conexão com o controlador. Tal comunicação pode ou não utilizar um canal seguro de comunicação (usualmente implementado por TCP ou TLS). Esses dispositivos podem ser tanto físicos quanto virtuais, como no caso do Open vSwitch (OVS, 2014).

Essa composição da rede facilita a introdução de novas funcionalidades, uma vez que toda lógica de controle é centralizada e não precisa ser implementada nos dispositivos de encaminhamento (como é feito no modelo tradicional). Apesar disso, SDN carece de suporte à programação ao nível de aplicação, essencial para a criação de funcionalidades mais complexas.

2.2 Regras em nível de aplicação

Atualmente, o processamento de pacotes ao nível da camada de aplicação é realizado através de *middleboxes* disponibilizados por empresas especializadas. Tais componentes possuem APIs proprietárias incompatíveis entre si. Além disso, eles são fabricados para uma solução geral, necessitando que o usuário de tais produtos adapte suas necessidades às funcionalidades providas (QAZI et al., 2013). Dessa forma, uma extensão natural da arquitetura SDN seria a inserção de suporte a regras em nível de aplicação, possibilitando a criação de funcionalidades mais complexas de forma facilitada. Esta seção discute os principais desafios e propostas em relação ao suporte de regras em nível de aplicação em arquiteturas SDN.

2.2.1 Desafios

O principal desafio é a falta de uma estrutura fixa nos cabeçalhos da camada de aplicação. A versão atual do OpenFlow [1], em linha com o modelo de programação de fluxo SDN, permite a definição de regras a partir de campos dos cabeçalhos das camadas de Enlace, de Rede e de Transporte. Tais campos possuem protocolos bem definidos e que podem ser encontrados a partir de um *offset* conhecido no pacote. Na camada de aplicação, por sua vez, os protocolos muitas vezes não possuem cabeçalhos estáticos, com tamanhos bem definidos ou uma ordem para a localização dos campos. Por exemplo, no HTTP, os valores de cada campo não possuem tamanho fixo; além disso, os campos após a primeira linha não são obrigatórios e a ordem em que aparecem é arbitrária. Essa característica acaba por dificultar o reconhecimento dos protocolos e seus campos em dispositivos de encaminhamento.

Além da falta de estrutura fixa, outros aspectos dificultam o reconhecimento de protocolos da camada de aplicação. O conteúdo das mensagens pode ser fragmentado pelas camadas inferiores em pacotes de tamanhos menores. Isso dificulta ainda mais o reconhecimento, uma vez que partes do cabeçalho da aplicação podem estar separadas em múltiplos pacotes. Além disso, a criptografia impossibilita o acesso ao *payload* do pacote, incluindo cabeçalhos de aplicações.

Com isso, a complexidade por esses desafios acarreta inexistência de regras em nível de camada de aplicação no protocolo OpenFlow e em arquiteturas SDNs atuais. Nesse contexto, este artigo propõe um estudo precursor sobre a viabilidade, complexidade e sobrecarga impostos na implementação tal suporte em uma arquitetura SDN OpenFlow.

2.2.2 Propostas

No conhecimento dos autores deste trabalho, a literatura atual não apresenta qualquer proposta com suporte a criação de regras em nível de aplicação. No entanto, foram encontrados dois estudos que, de certa forma, realizam processamento em nível de aplicação, sem a definição de regras específicas para isso.

O framework AVANT-GUARD (SHIN et al., 2013) adiciona funcionalidades ao plano de dados OpenFlow para tratar problemas de segurança em uma rede SDN. Ele protege o plano de controle contra ataques de saturação gerados no plano de dados com *TCP-SYN flood*. Para isso são utilizados dois módulos. O primeiro é uma migração de conexão, que faz a inspeção das sessões TCP no plano de dados antes de notificar o controlador. O segundo módulo, chamado de *trigger* atuador, habilita o controlador a instalar regras no plano de dados para detectar e responder a ameaças observadas. Como limitação, a

[1] Durante a escrita deste documento, a versão mais recente era a 1.4 (ONF, 2013))

proposta suporta somente proteção a ataques do tipo *TCP-SYN flood* e cria uma sobrecarga devido a migração de conexão.

Application-aware Data Plane Processing in SDN (MEKKY et al., 2014) busca adicionar ao Openflow suporte a ações que utilizam informação contidas na camada de aplicação. Para isso, foi criado um módulo no dispositivo de encaminhamento para compilar, instalar e executar ações de *middleboxes*. Tal módulo opera como uma espécie de controlador. Primeiramente, ao ocorrer um *table-miss*, o pacote é redirecionado ao módulo e analisado sobre as regras ali instaladas. Essas regras são instaladas pelo controlador e são da forma <*matches*, *applications*>, onde *matches* corresponde aos campos L2-L4 (como no padrão OpenFlow) e *applications* corresponde às aplicações pré-compiladas no *switch* (ex.: *Firewall*, *Load Balancer*, etc). Caso ocorra uma correspondência com alguma regra, as aplicações da lista são aplicadas, na ordem especificada. Todavia, caso não ocorra, o pacote é encaminhado ao controlador real. O trabalho demonstra a aplicabilidade com um estudo de caso. A principal limitação, contudo, é a alta sobrecarga na ocorrência de um table-miss, uma vez que qualquer pacote sofre um processamento preliminar antes de ser encaminhado ao controlador.

3 APPFLOW

Este capítulo apresenta a proposta AppFlow. A Seção 3.1 descreve o projeto de extensão da arquitetura SDN, enquanto a Seção 3.2 discute a implementação do protótipo orientado a HTTP.

3.1 Visão Geral

Para incorporar regras em nível de aplicação na arquitetura SDN OpenFlow, foram criadas extensões em cada camada da arquitetura padrão, conforme ilustrado na Figura 3.2. O objetivo do projeto é que a separação entre as camadas seja mantida, de forma que as regras sejam definidas no Serviço de Controle, enviadas para o Controlador e instaladas e executadas nos Dispositivos de Encaminhamento. A Figura 3.1 demonstra a sequência de operações do AppFlow. O funcionamento inicia quando o Serviço de Controle envia (E1) para o Controlador as regras de camada de aplicação. O Controlador, por sua vez, armazena as regras em uma estrutura e, a seguir, sinaliza (E2) o Dispositivo de Encaminhamento sobre os protocolos que devem ser monitorados. Ao receber essa sinalização, o Dispositivo de Encaminhamento inicia um monitoramento dos pacotes a fim de encontrar os protocolos sinalizados. Quando um protocolo é identificado em um pacote, o Dispositivo de Encaminhamento envia-o (E3) para o Controlador. Quando isso ocorre, o Controlador instala uma regra (E4) no Dispositivo de Encaminhamento especificando o fluxo ao qual o pacote pertence, utilizando, para isso, informações de camadas inferiores. Segue uma breve explicação sobre a mudança necessária em cada elemento.

Serviços de Controle. Novo padrão de definição de regras em nível de camada de aplicação, definido por AppFlow. O modelo adotado para a definição de regras é a tupla <Protocolo, Chave, Valor, Ação> (ex.: regra que bloqueia o host inf.ufrgs.br — <HTTP, host, inf.ufrgs.br, drop>). Esses quatro campos permitem que políticas de rede sejam programadas de forma simples. O administrador da rede precisa apenas de conhecimentos básicos de redes de computadores; ele não necessita entender as especifidades de SDN ou da extensão AppFlow.

API Norte. Interface que o controlador disponibiliza para a aplicação. Nesse nível tem-se a especificação dos protocolos, chaves, valores e ações suportados pela implementação.

Controlador. Tradução da regra alto nível (camada de aplicação) para a estrutura OpenFlow de instalação de regras. O modelo atual assume duas premissas em relação a esta etapa. Primeiro, o próprio dispositivo de encaminhamento reconhece pacotes da camada de aplicação e vincula-os a fluxos definidos pelas camadas inferiores. Segundo, regras da camada de aplicação podem ser traduzidas para regras que utilizem as camadas mais abaixo. O objetivo dessas premissas é manter o funcionamento da arquitetura

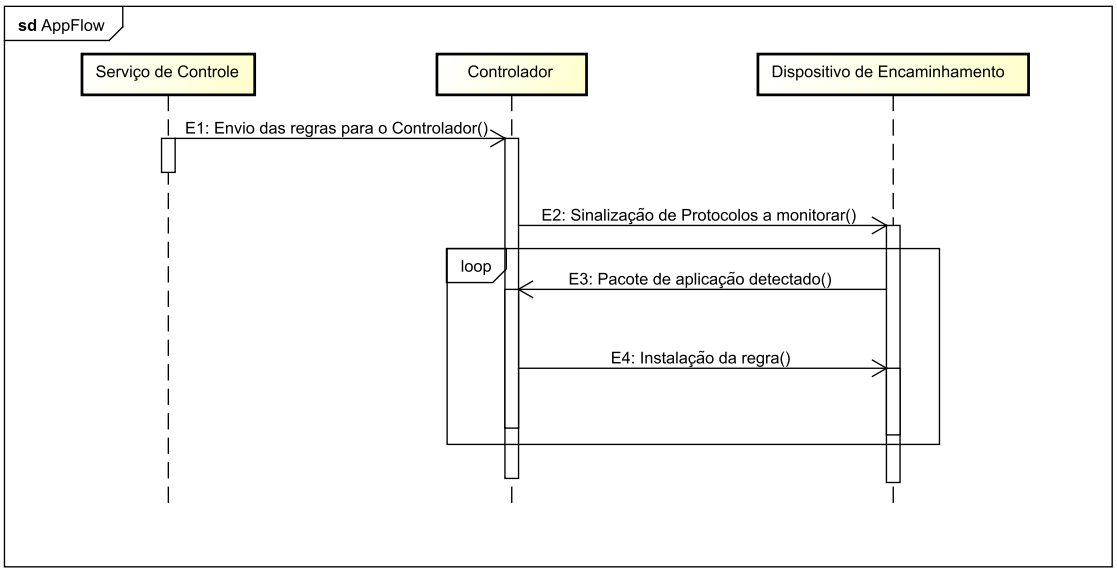

Figura 3.1: Diagrama de sequência do AppFlow.

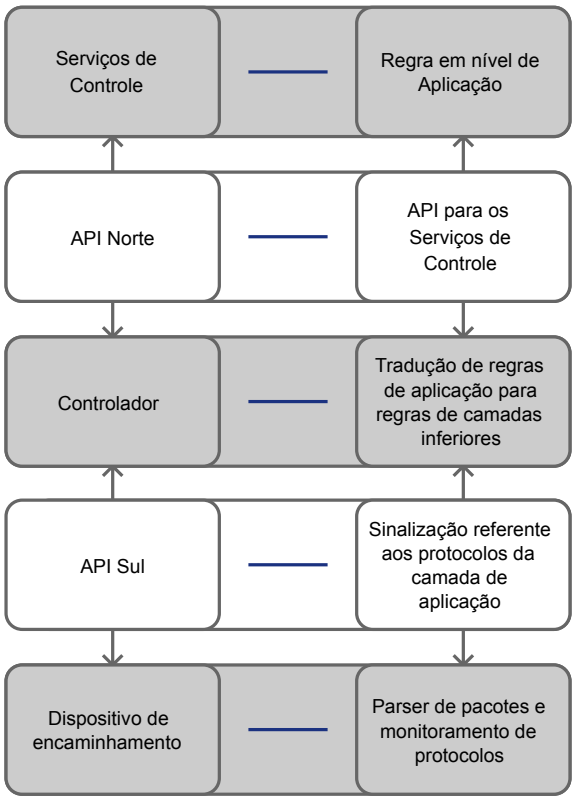

Figura 3.2: Camadas da arquitetura SDN OpenFlow e mudanças propostas em AppFlow.

próximo ao original. O funcionamento é dado da seguinte forma: ao reconhecer um pacote como pertencente a algum protocolo de aplicação de interesse, o pacote é analisado e as ações correspondentes a esse fluxo são atualizadas. Assim, as ações são aplicadas a todos os pacotes do mesmo fluxo, sem a necessidade de que haja análise de cada pacote subsequente.

API Sul. Interface que realiza a comunicação entre o controlador e o dispositivo de encaminhamento. Aqui, tem-se a especificação dos protocolos da camada de aplicação que deverão ser monitorados pelo dispositivo de encaminhamento e o reconhecimento enviado pelo dispositivo ao identificar um pacote. Como o modelo não é restrito a um formato específico de aplicação, essa informação deverá ser um tipo de identificação para que o dispositivo de encaminhamento saiba a qual protocolo se refere. O objetivo desse modelo de API é para que tenha-se flexibilidade para incorporar novas aplicações ao AppFlow.

Dispositivo de Encaminhamento. Tem-se como requisitos realizar as modificações necessárias para que o dispositivo de encaminhamento instale a regra nas tabelas de fluxo e, também, a criação do mecanismo que realizará o *match* nos pacotes.

A partir desse modelo foram tomadas as decisões de implementação que serão discutidas a seguir.

3.2 Protótipo

Visto que o conjunto de protocolos da camada de aplicação é amplo e de difícil generalização, foi implementado um protótipo para prova de conceito. Esta seção apresenta as decisões de projeto e a implementação do protótipo, comparando as diferenças com o TG1, quando necessário.

3.2.1 Decisões de Projeto

As decisões de implementação foram feitas de acordo com Tabela 3.1. A tabela compara as especificações feitas no TG1 com as implementadas no TG2. Segue uma explicação sobre cada camada da arquitetura AppFlow.

Camadas da Arquitetura	TG1	TG2
Serviços de Controle	Não especificado	HTTP
API Norte	Dependente do Controlador	Independente do Controlador
Controlador	RYU	RYU
API Sul	OpenFlow 1.3	OpenFlow 1.3
Dispositivo de encaminhamento	Open vSwitch	OFSoftSwitch13

Tabela 3.1: Decisões de implementação nos elementos da arquitetura SDN OpenFlow.

Serviços de Controle. No TG1 não houve especificação sobre qual protocolo da camada de aplicação a ser utilizado no protótipo. No TG2 foi escolhido o HTTP para prova de conceito, uma vez que é um protocolo de especificação aberta e amplamente utilizado.

API Norte. No TG1, a API Norte seria vinculada ao código e à linguagem do Controlador, sendo assim dependente da escolha do mesmo. No TG2 a implementação foi desvinculada do Controlador para simplificar a criação dos Serviços de Controle. Para isso, a API passou a ser uma interface para importação de regras em nível de aplicação de um arquivo de texto.

Controlador. Em ambos os casos, a escolha foi o RYU. Isso se deve ao fato de que o RYU é o único controlador aberto com suporte à última versão do OpenFlow.

API Sul. Em ambos os casos, a escolha foi o OpenFlow 1.3. A versão está vinculada ao Dispositivo de Encaminhamento escolhido.

Dispositivo de Encaminhamento. Houve mudança em relação ao TG1. Passou-se a utilizar o OFSoftswitch13 (CPqD, 2014) ao invés do Open vSwitch (OVS, 2014). Tal mudança ocorreu devida à simplificação substancial na implementação oferecida pelo OFSoftSwitch. O OFSoftSwitch13 oferece suporte nativo a operações na carga útil de pacotes e o monitoramento de protocolos da camada de aplicação. No Open vSwitch, em contraste, tais operações demandam diversas modificações no código do *switch*, nas bibliotecas de apoio utilizadas pelo mesmo e no *kernel* do Linux.

3.2.2 Implementação

Em linha com o que foi descrito na subseção anterior, na implementação do projeto as seguintes mudanças foram realizadas em cada camada da arquitetura.

Serviços de Controle. O administrador de rede define seu Serviço de Controle por meio de regras HTTP da seguinte forma: <Chave, Valor, Ação>, aqui a tupla difere da especificada na Seção 3.1 devido ao suporte ser só para o protocolo HTTP (o que torna desnecessária a chave Protocolo). A Chave pode ser definida por qualquer chave compatível com o protocolo HTTP (ex.: *method*, *url*, *host*, *version* etc. O Valor deve ser algum valor esperado pela chave definida no campo anterior. Por sua vez, a Ação deve ser alguma ação definida pelo padrão OpenFlow (ex.: descartar, encaminhar para porta, flooding etc).

API Norte. O protótipo utiliza um arquivo de texto com uma lista de regras <Chave, Valor, Ação>, dispostas uma por linha. Não há limite especificado para o número de regras. O arquivo texto serve de interface entre o Serviço de Controle e o Controlador para a definição das regras em nível de aplicação. Segue um exemplo de arquivo de texto, no qual a primeira regra bloqueia o *host* facebook.com, a segunda redireciona todos os pacotes HTTP com método PUT para a porta 1 e, a terceira regra descarta todos os pacotes HTTP com versão 1.0.

```
host, facebook.com, drop
method, PUT, output 1
version, 1.0, drop
```

Controlador. Ao ser inicializado, o Controlador carrega o arquivo com as definições de regras e armazena em uma estrutura interna. A etapa de enviar ordens de monitoramento HTTP aos Dispositivos de Encaminhamento é ignorada. Tal monitoramento é assumido por padrão, pois o protótipo é específico para o protocolo HTTP. Ao receber um pacote HTTP, o controlador realiza uma busca na carga útil e, caso encontre alguma das triplas definidas no arquivo, o controlador instala, no *switch*, uma regra baseada no endereço IP e porta TCP destino do pacote com a ação definida na tripla.

API Sul. Não foram realizadas alterações, uma vez que todas as ações necessárias já são comportadas pelo protocolo OpenFlow.

Dispositivo de Encaminhamento. No código do dispositivo de encaminhamento foi necessário adicionar o reconhecimento do protocolo HTTP e o envio do pacote para o controlador, no caso de reconhecimento positivo.

Para o controle padrão da rede, utilizou-se como base uma aplicação de *switch* Ethernet simples e adicionou-se a inteligência necessária para o tratamento de pacotes HTTPs.

Para adicionar o reconhecimento do protocolo HTTP foi utilizada a biblioteca Netbee (NetBee, 2014), já inclusa no OFSoftswitch13. Essa biblioteca utiliza uma linguagem chamada NetPDL, que define, através de um XML, especificações de protocolos das camadas 2 a 7. Dessa maneira, foi adicionado o código correspondente ao protocolo HTTP à especificação NetPDL do *switch*. No código do *switch* adicionou-se o tratamento necessário para que, no caso de reconhecimento positivo, o pacote fosse enviado para o controlador dentro de uma mensagem padrão OpenFlow. A Listagem 3.1 contém um trecho, correspondente ao reconhecimento HTTP, da biblioteca NetPDL [1].

```
<if expr="($packet[$currentoffset : 3] == 'GET') or ($packet[$currentoffset : 4] == '
    POST') or ($packet[$currentoffset : 4] == 'HTTP')">
 <if-true>
  <block name="header" longname="HTTP Header">

    <!-- Requestline and statusline-->
    <if expr="($packet[$currentoffset : 3] == 'GET') or ($packet[$currentoffset : 4] ==
        'POST')">
     <if-true>
      <field type="line" name="cmdline" longname="Command Line" showtemplate="
          FieldAscii">
       <field type="tokenended" name="method" longname="Method" endtoken=" "
           showtemplate="FieldAscii"/>
       <field type="tokenended" name="url" longname="URL" endtoken=" " showtemplate="
           FieldAscii"/>
       <field type="line" name="reqVersion" longname="Version" showtemplate="
           FieldAscii"/>
      </field>
     </if-true>

     <if-false>
      <field type="line" name="statusline" longname="Status Line" showtemplate="
          FieldAscii">
       <field type="tokenended" name="repVersion" longname="Version" endtoken=" "
           showtemplate="FieldAscii"/>
       <field type="tokenended" name="statuscode" longname="Status Code" endtoken=" "
           showtemplate="FieldAscii"/>
       <field type="line" name="reasonphrase" longname="Reason Phrase" showtemplate="
           FieldAscii"/>
      </field>
     </if-false>
    </if>
```

Listagem 3.1: Trecho arquivo biblioteca NetPDL

Assim, após o carregamento do arquivo com as triplas definidas pelo usuário, o Controlador entra no modo de funcionamento de um *switch* Ethernet padrão. Quando houver recebimento de pacote com carga útil HTTP, ele verificará se algum item do arquivo corresponde com algum campo do pacote e, no caso positivo, será instalada uma regra no *switch*. Essa regra terá *match* baseado no endereço IP Destino com a ação especificada pelo usuário. Nota-se que nessa implementação utilizou-se o protocolo HTTP. Entretanto, caso fosse desejado expandir o protótipo para algum outro protocolo, bastaria que seu reconhecimento fosse adicionado à especificação NetPDL no OFSoftSwitch13 e seu conjunto de regras habilitado junto ao arquivo de texto — definido na API Norte.

[1] O arquivo completo referente ao reconhecimento HTTP pode ser encontrado em: http://www.nbee.org/netpdl/wiki/http

4 AVALIAÇÃO

Neste capítulo, na Seção 4.1 são descritos os cenários de avaliação, as métricas e os fatores. Na Seção 4.2, apresenta-se o ambiente utilizado para a realização da avaliação, o que inclui a plataforma de simulação e as ferramentas de medição. Por fim, na Seção 4.3, são discutidos os resultados obtidos.

4.1 Metodologia

A avaliação é guiada por três questões fundamentais. A seguir são apresentadas as questões e são descritos os cenários, métricas e fatores utilizados para respondê-las.

- **Q1:** O plano de dados monitora corretamente o protocolo da camada de aplicação especificado pelo plano de controle?
- **Q2:** As ações especificadas no plano de controle estão sendo corretamente aplicadas pelo plano de dados?
- **Q3:** Qual o impacto no desempenho, ocasionado pela sobrecarga adicionada?

Cenário 1: Monitoramento HTTP. Para responder a questão 1 utilizou-se um servidor HTTP local recebendo requisições de um *host* externo. Essas requisições passam pelo *switch* com reconhecimento HTTP ligado. O objetivo esperado é que todas as requisições sejam identificadas.

Cenário 2: Controle parental. Esse cenário visa a responder a questão 2. Nele, *websites* são bloqueados por meio de regras definidas com o Serviço de Controle AppFlow. São utilizados *websites* reais e um *host* externo conectado a Internet por meio de um Dispositivo AppFlow. Todas as requisições para os websites bloqueados devem ser descartadas pelo *switch*.

Cenário 3: *Benchmark*. O objetivo deste cenário é responder a terceira questão. Dois *hosts*, conectados por meio de um *switch* AppFlow, utilizam ferramentas para medição da qualidade de tráfego entre eles. São utilizadas ferramentas padrão de medição de desempenho em redes de computadores (a saber, Iperf, SockPerf e Ping). As configurações específicas de cada ferramenta são descritas na próxima subseção.

Métricas. São utilizadas quatro métricas para os cenários 1 e 3: (i) o número absoluto de requisições HTTP; (ii) a taxa de requisições HTTP por segundo; (iii) a vazão entre *hosts* (TCP, UDP); e (iv) latência entre *hosts*.

Fatores. Existem dois fatores principais que afetam o desempenho do OFSoftswitch13. O primeiro fator é o Modo de Inspeção de Pacotes. Este fator possui dois valores: padrão do *switch* (inspeciona os cabeçalhos L2 a L4); ou HTTP (estende o padrão para buscar

cabeçalhos HTTP na carga útil dos pacotes). O segundo fator é uma *flag* de Conversão, a qual indica se o conteúdo do pacote deve, ou não, ser convertido para *strings*. Por padrão a *flag* é desabilitada no *switch*, indicando que os valores dos pacotes serão analisados em binário. No entanto, para a verificação correta de pacotes HTTP é necessário habilitar a *flag*, convertendo o conteúdo para *strings*. O conjunto total de combinações de configuração está resumido na Tabela 4.1. As configurações funcionais são a 1 (*switch* padrão) e a 4 (*switch* HTTP), as intermediárias servem para se avaliar o efeito de cada fator.

Configurações	Modo de Inspeção	Conversão
1	Padrão	Desabilitada
2	Padrão	Habilitada
3	HTTP	Desabilitada
4	HTTP	Habilitada

Tabela 4.1: Combinações de fatores nos experimentos.

4.2 Ambiente

Esta seção descreve os equipamentos, a topologia, a configuração e as ferramentas utilizados nos experimentos. Todos os experimentos foram realizados em um mesmo computador com processador Intel i7 (dois núcleos físicos de 1.9Ghz), 12GB memória RAM e sistema operacional Windows 8.1. Os cenários de avaliação foram realizados sobre uma topologia virtual criada com o simulador GNS3 (GNS3 Technologies Inc, 2014). A topologia consiste em quatro máquinas virtuais, dois *hosts*, um *switch* e um controlador, interconectados conforme observado na Figura 4.1.

Os experimentos foram executados da seguinte forma. Nos Cenários 1 e 3, o *Host* H1 envia tráfego para o *Host* H2 por meio do *switch* AppFlow. No Cenário 2, o tráfego ocorre entre o *Host* H1 e a Internet. Conforme discutido anteriormente, o objetivo da avaliação é uma primeira análise da viabilidade da proposta. Portanto, apesar de simples, a topologia e os cenários são suficientemente representativos para tal objetivo. O desempenho em larga escala da solução será avaliado em trabalhos futuros.

Por fim, para executar os experimentos foram utilizadas as seguintes ferramentas:

- **Iperf**: ferramenta para geração de tráfego, configurada em ambos os modos TCP e UDP;
- **sockperf**: ferramenta com um conjunto de funcionalidades para avaliação de desempenho; configurada em modo *ping-pong*, ou seja, para medir a latência entre dois *hosts* de cada pacote a uma resolução de nanosegundos;
- **ping**: ferramenta para verificação de conectividade e RTT via ICMP; configurada em modo *flood*, ou seja, para enviar requisições intermitentemente durante um período definido;
- **http-server** (node-js, 2014): servidor HTTP 1.1 simples com suporte a múltiplas requisições paralelas;
- **wget**: ferramenta simples para requisições de conteúdo HTTP.

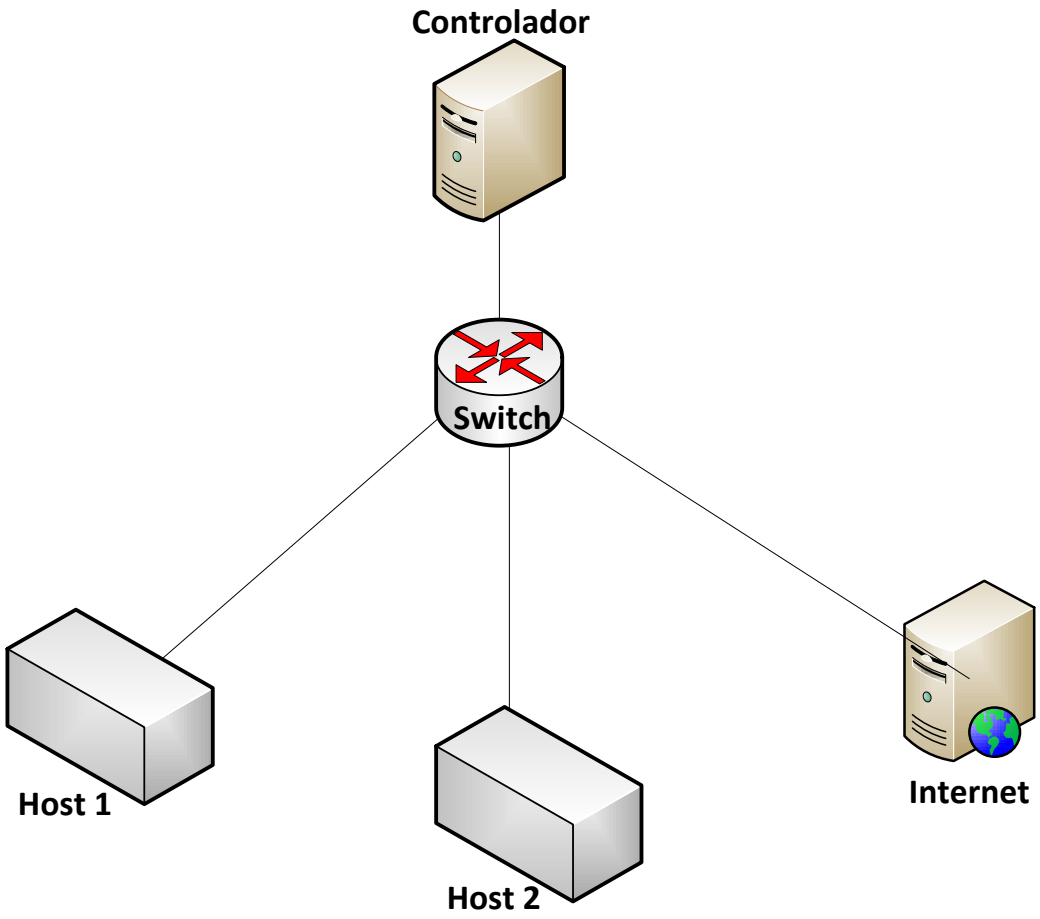

Figura 4.1: Topologia utilizada para a realização dos experimentos.

4.3 Resultados

Nesta seção serão apresentados os resultados obtidos com os experimentos executados nos três cenários propostos (Seção 4.1):

Monitoramento HTTP. Nesse cenário são geradas requisições HTTP concorrentes, com wget, para o http-server. Primeiramente, através de um estudo de sensibilidade, verificou-se que a taxa máxima de requisições foi atingida com 20 execuções wget em paralelo. Em seguida, foram realizadas duas variações do experimento: uma sem qualquer tráfego de fundo; e a outra, com um tráfego UDP de fundo enviando à taxa máxima do enlace. Cada experimento foi repetido 30 vezes.

Para o primeiro caso, foram identificadas, na média 77 requisições por segundo. Para o segundo caso foram 26 requisições por segundo. Em todos os experimentos, a taxa de acerto foi de 100%. Isso ocorre porque o monitoramento HTTP é realizado pelo *switch* em todos os pacotes. As implicações desse monitoramento por pacotes serão discutidas no terceiro cenário (*benchmark*).

Controle Parental. No cenário 2, o administrador da rede adiciona no arquivo uma regra do tipo <host, example.com, drop>. Essa regra irá fazer o *switch* bloquear todos os pacotes direcionados para o servidor <example.com>, descartando, para isso, todos os pacotes cujo IP destino remete ao servidor.

O descarte ocorre apenas após a identificação do valor "example.com" no campo "host" dos pacotes HTTP. Desse modo, durante a primeira tentativa de comunicação, os pacotes referentes ao *Three-Way Handshake* do TCP são encaminhados normalmente. Após a identificação do protocolo HTTP, do campo "host" e do valor "example.com", o endereço do servidor é bloqueado e os pacotes subsequentes são descartados. Dessa maneira, novas tentativas de conexão serão descartadas antes do *Three-Way Handshake*. Por outro lado, se o servidor possui múltiplos endereços IP atribuídos, cada endereço será adicionado individualmente à lista de bloqueio durante a primeira tentativa de conexão TCP e todos serão bloqueados. Nesses experimentos, todos os pacotes subsequentes à primeira identificação do servidor são descartados. Isso ocorre porque, de forma similar aos experimentos do Cenário 1, todos os pacotes são verificados no *switch*. O próximo cenário analisa as implicações dessa escolha no desempenho da solução.

Benchmark. Por fim, nesse cenário foram realizadas quatro medições (duas de vazão e duas de latência) sobre as configurações possíveis de fatores, descritas na Seção 4.1 (Tabela 4.1). Cada gráfico representa uma medição (de vazão ou latência). Em cada gráfico são traçados quatro valores, os quais correspondem às configurações de fatores. Ademais, em todos os casos foram executadas 30 repetições. Os valores observados são as médias e os desvios padrão relacionados às repetições.

As Figuras 4.2 e 4.3 demonstram a vazão, em megabits por segundo, de tráfegos gerados entre os *hosts* H1 e H2. Nos experimentos com tráfegos TCP (Figura 4.2), o fluxo é mantido por tempo suficiente para que o valor da vazão possa convergir. Nos experimentos com UDP, foi realizado um teste de sensibilidade em cada configuração, com busca binária, para se descobrir o valor máximo de envio. Nesse caso, foram analisadas as taxas de perdas de pacotes informadas pelo Iperf. Foi adotado o maior valor de taxa de envio que gerou 0% de perdas. Após a calibragem do sistema foram executadas as variações e repetições.

Conforme é possível observar, existe variação no desempenho em todas as configurações de fatores. A maior queda de desempenho ocorre na habilitação do Modo de Inspeção com HTTP (configurações 3 e 4). Para o tráfego TCP (Figura 4.2), a perda relativa de desempenho, nesse caso, foi de 71,77% sem conversão para *strings* (entre as configu-

rações 1 e 3) e de 67,04% com conversão para *strings* (entre as configurações 2 e 4). Em relação ao UDP (Figura 4.3), os valores foram de 56,71% e 60,02%, respectivamente.

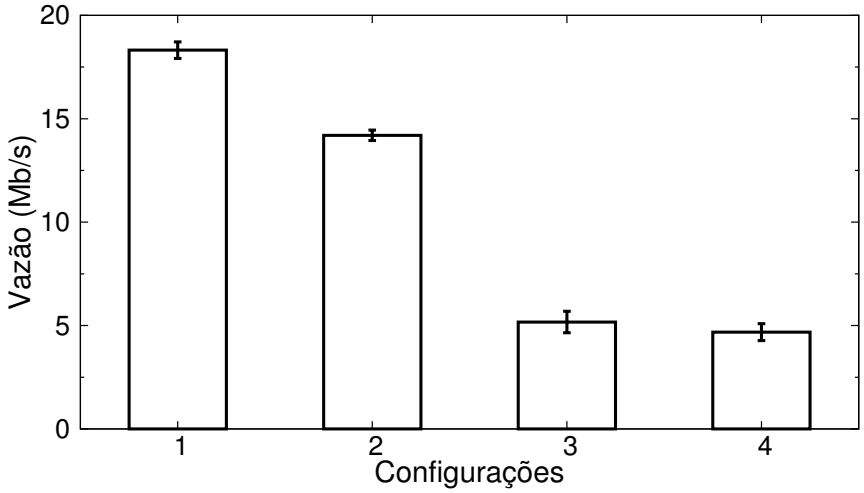

Figura 4.2: Vazão medida com Iperf modo TCP.

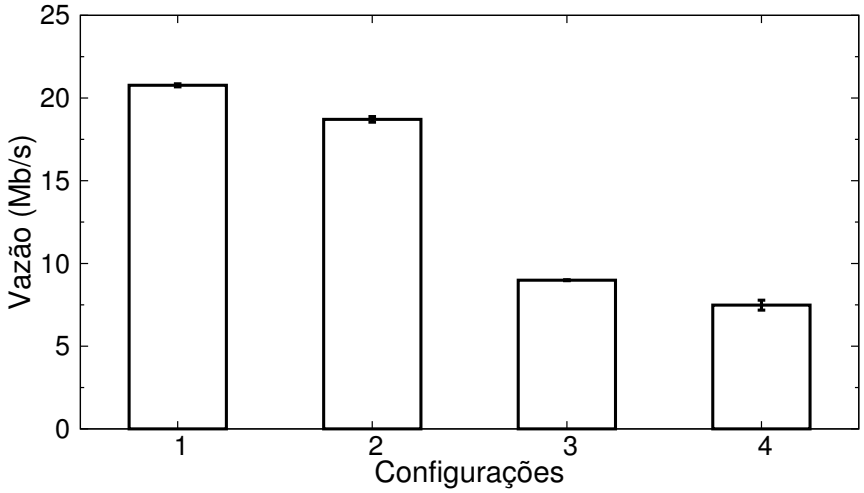

Figura 4.3: Vazão medida com Iperf modo UDP.

Para medida de latência temos as Figuras 4.5 e 4.4, onde a primeira mede o RTT e a segunda o RTT médio dividido por dois. Nos experimentos realizados com a ferramenta Ping (Figura 4.5), foi realizado um teste de sensibilidade, com busca binária, para se descobrir qual o tempo mínimo de execução a fim de se ter um valor médio de RTT que convergia nas execuções. Com a ferramenta SockPerf, utilizou-se a mesma abordagem para se obter o tempo mínimo de execução. Após a descoberta desse tempo, foram executadas as variações e repetições.

Como é possível observar e já mostrado nos experimentos de medida de Vazão, houve um aumento na latência do sistema (queda de desempenho). Da mesma forma que nos experimentos de Vazão, a maior queda ocorreu na habilitação do Modo de Inspeção HTTP. Assim, para os experimentos de latência — medidos com RTT (ping) (Figura 4.5 — obteve-se um aumento na latência de 11,11% sem conversão para *strings* e de 3.12% com conversão para *strings*. Já nas medidas de latência — medidas com Sockperf 4.4 — os valores foram de 11.65% e 3.68%, respectivamente. A lição dessas avaliações é a de que

o custo de processamento HTTP no *switch* pode ser proibitivo, mesmo quando a função é apenas a de monitoramento e sinalização.

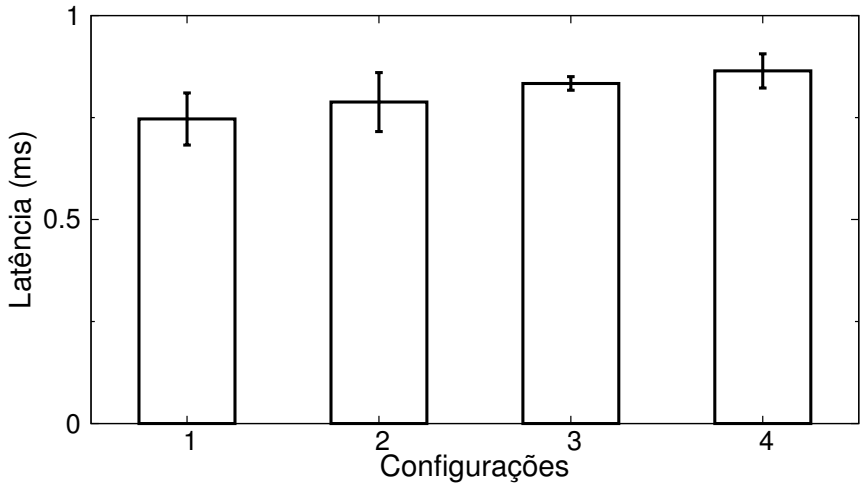

Figura 4.4: Latência medida com a ferramenta SockPerf.

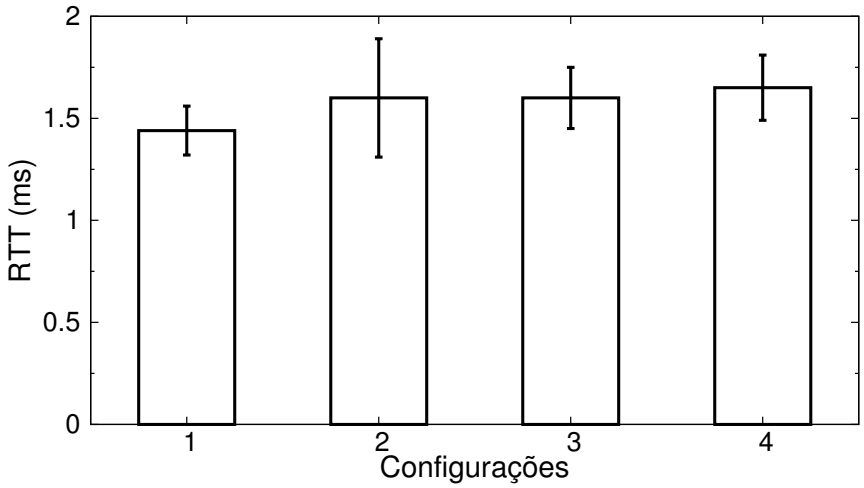

Figura 4.5: *Round-trip time* medido com a ferramenta Ping.

5 CONCLUSÃO

Redes Definidas por Software (SDN) surgiram como uma oportunidade de resolver alguns dos problemas que se têm nas redes de computadores, atualmente, como a alta complexidade e a dificuldade de administrar. Um dos motivos disso é a alta integração vertical entre o plano de dados e o plano de controle. SDN visa a solucionar esse problema através da desassociação entre o plano de dados e o plano de controle, em que o primeiro consiste em simples comutadores de pacotes programados remotamente por um controlador centralizado. Dessa maneira, SDN representa uma significativa mudança de paradigma no desenvolvimento e na evolução das redes de computadores.

Neste trabalho, estudou-se a possibilidade de avançar o paradigma SDN, através de uma extensão para seu protocolo mais utilizado atualmente, o OpenFlow. Essa extensão tem por finalidade criar um protótipo que habilite a criação de regras em nível de aplicação em um *switch* compatível com OpenFlow. Devido a desafios percebidos durante a implementação do projeto, a ideia inicial foi simplificada e o suporte ficou limitado ao protocolo de aplicação HTTP. Apesar disso, o protótipo ficou funcional e pode-se validar sua viabilidade através de experimentos realizados.

Por um lado, os experimentos comprovaram que o protótipo funcionava como esperado. Por outro, os resultados não foram muito promissores em relação ao desempenho. Utilizando ferramentas de *benckmark*, obteve-se um decréscimo de 63% na vazão e um aumento de 15% na latência, em comparação ao *switch* padrão, no ambiente analisado. Dado que os resultados foram preliminares, devido à configuração do ambiente, justifica-se uma outra avaliação para que se possa afirmar que o processamento em nível de aplicação é, ou não, viável junto aos Dispositivos de Encaminhamento.

Como trabalhos futuros, serão avaliadas outras implementações de *switches* e a avaliação em topologia realística, para comparação dos resultados e obtenção de novos fatores para investigação. De toda forma, apesar de os resultados não terem sidos muito favoráveis, o trabalho serviu como prova de conceito para demonstrar a dificuldade de se ter o suporte a regras em nível de aplicação diretamente nos *switches* com o protocolo OpenFlow.

REFERÊNCIAS

NOXRepo.org . **Nox [Online]**. Acesso em: 19 dez. 2014, Disponível em <http://www.noxrepo.org/nox/about-nox/>.

NOXRepo.org . **POX [Online]**. Acesso em: 19 dez. 2014, Disponível em <http://www.noxrepo.org/nox/about-nox/>.

Big Switch Networks. **Project Floodlight [Online]**. Acesso em: 19 dez. 2014, Disponível em <http://www.projectfloodlight.org/floodlight/>.

CPqD. **OpenFlow 1.3 Software Switch [Online]**. Acesso em: 10 nov. 2014, Disponível em <https://github.com/CPqD/ofsoftswitch13>.

GNS3 Technologies Inc. **GNS3 [Online]**. Acesso em: 27 nov. 2014, Disponível em <http://www.gns3.com/>.

JARRAYA, Y.; MADI, T.; DEBBABI, M. A Survey and a Layered Taxonomy of Software-Defined Networking. **Communications Surveys Tutorials, IEEE**, [S.l.], v.PP, n.99, p.1–1, 2014.

KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-Defined Networking: A comprehensive survey. **CoRR**, [S.l.], v.abs/1406.0440, 2014.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network Innovation using OpenFlow: a survey. **Communications Surveys Tutorials, IEEE**, [S.l.], v.16, n.1, p.493–512, First 2014.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. OpenFlow: enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, New York, NY, USA, v.38, n.2, p.69–74, Mar. 2008.

MEKKY, H.; HAO, F.; MUKHERJEE, S.; ZHANG, Z.-L.; LAKSHMAN, T. Application-aware Data Plane Processing in SDN. In: THIRD WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING, New York, NY, USA. **Proceedings** ACM, 2014. p.13–18. (HotSDN '14).

NetBee. **NetBee Library**. Acesso em: 24 nov. 2014, Disponível em <http://www.nbee.org/doku.php>.

node-js. **NPM node.js http-server**. Acesso em: 25 nov. 2014, Disponível em <https://www.npmjs.org/package/http-server>.

NUNES, B.; MENDONCA, M.; NGUYEN, X.; OBRACZKA, K.; TURLETTI, T. A Survey of Software-Defined Networking: past, present, and future of programmable networks. **Communications Surveys Tutorials, IEEE**, [S.l.], v.PP, n.99, p.1–18, 2014.

ONF. **OpenFlow Switch specification Version 1.4.0**. Acesso em: 5 jun. 2014, Disponível em <http://goo.gl/aBAo5T>.

ONF. **SDN Architecture**. Acesso em: 12 nov. 2014, Disponível em <http://goo.gl/yhvG1a>.

ONF. **Open Networking Foundation [Online]**. Acesso em: 12 nov. 2014, Disponível em <http://www.opennetworking.org>.

OVS. **Open vSwitch - An Open Virtual Switch**. Acesso em: 12 nov. 2014, Disponível em <http://openvswitch.org>.

QAZI, Z. A.; TU, C.-C.; CHIANG, L.; MIAO, R.; SEKAR, V.; YU, M. SIMPLE-fying Middlebox Policy Enforcement Using SDN. **SIGCOMM Comput. Commun. Rev.**, New York, NY, USA, v.43, n.4, p.27–38, Aug. 2013.

Ryu. **Ryu SDN Framework [Online]**. Acesso em: 6 jun. 2014, Disponível em <http://osrg.github.io/ryu/>.

SHIN, S.; YEGNESWARAN, V.; PORRAS, P.; GU, G. AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In: ACM SIGSAC CONFERENCE ON COMPUTER & COMMUNICATIONS SECURITY, 2013., New York, NY, USA. **Proceedings** ACM, 2013. p.413–424. (CCS '13).

ANEXO A: TRABALHO DE GRADUAÇÃO I

AppFlow: Explorando Suporte a Regras em Nível de Aplicação em OpenFlow

Gustavo Miotto[1], **Marinho P. Barcellos**[1]

[1]Instituto de Informática - Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500 - Porto Alegre, RS - Brasil

{gmiotto, marinho}@inf.ufrgs.br

Abstract. *Software-defined networking (SDN) is a new paradigm which consists of decoupling the control and data plane of a network, so that a central logic controller configures the forwarding elements according to the programs it runs. Such a separation was created aiming to overcome the problems we currently have, for example, the difficult to evolve and managing the network. OpenFlow is the protocol widely adopted by the community to implement SDN. Despite the progress already achieved, there is still plenty to evolve until the OpenFlow get the maximum advantages that SDNs offer. This paper presents an extension to the current OpenFlow protocol, whose main objective is to enable features using rules based on application layer. The present study will serve as a basis for the realization of Graduation Work 2.*

Resumo. *Redes definidas por software (SDN) é um novo paradigma em que temos a separação do plano de dados do plano de controle de uma rede, de maneira que um controlador lógico central configura os dispositivos de encaminhamento segundo os programas que ele executa. Tal separação foi criada visando superar os problemas que temos atualmente, por exemplo, a dificuldade de evoluir e administrar a rede. OpenFlow é o protocolo amplamente adotado pela comunidade para implementar SDN. Apesar dos avanços já obtidos, ainda há muito o que evoluir para que o OpenFlow possa obter o máximo de vantagens que SDNs oferecem. Este artigo apresenta uma proposta de extensão ao protocolo OpenFlow, cujo objetivo principal é possibilitar funcionalidades que utilizam regras baseadas na camada de aplicação. O estudo apresentado servirá de embasamento para a realização do Trabalho de Graduação 2.*

1. Introdução

Atualmente, redes de computadores são compostas por uma vasta gama de dispositivos, como roteadores, *switches* e *middleboxes*, que possuem muitos protocolos complexos implementados neles [Nunes et al. 2014]. Tal estrutura torna a administração, a configuração da rede e a implementação de novos protocolos uma tarefa desafiadora, pois os dispositivos podem ser de empresas diferentes – com interfaces diferentes – além de terem código proprietário (fechado).

Redes definidas por software (*Software-defined Networking* - SDN) trouxe à tona o conceito de redes programáveis, introduzindo um novo paradigma que promete simplificar drasticamente a administração da rede e prover inovação através da programabilidade da rede [Nunes et al. 2014]. A ideia principal por trás de SDN é a separação do plano

de dados do plano de controle da rede, de maneira que um controlador lógico central configura os dispositivos de encaminhamento segundo um programa que ele executa.

Devido às grandes possibilidades, as SDNs tem atraído muita atenção acadêmica e comercial. Em 2011, um grupo de empresas relacionadas com a área de redes se reuniu para formar a *Open Network Foundation* (ONF) [ONF 2014], tendo como objetivo principal promover SDNs e padronizar a comunicação entre os dispositivos de encaminhamento e o controlador, através do protocolo OpenFlow. O protocolo OpenFlow [McKeown et al. 2008], foi criado na Universidade de Stanford, com o objetivo inicial de prover uma plataforma que possibilitasse aos pesquisadores testar experimentos em redes de produção [Lara et al. 2014]. Sob o domínio da Open Networking Foundation [ONF 2014], o protocolo encontra-se atualmente na versão 1.4.0 [ONF 2013].

O protocolo OpenFlow [McKeown et al. 2008] é uma implementação que possibilita a configuração dos dispositivos de encaminhamento segundo o controlador central da rede SDN. Ele é utilizado para determinar o comportamento dos dispositivos de encaminhamento (DE) de acordo com a lógica desejada para a rede, definida via aplicações que executam no controlador. Portanto, seu funcionamento é baseado em um conjunto de mensagens que, interpretadas pelos DEs, são convertidas em regras de encaminhamento, que, por sua vez, são gravadas nas tabelas de fluxo dos DEs.

Apesar da área de estudo do protocolo estar em expansão, ainda existem muitos aspectos a serem explorados. Uma das limitações existentes é o suporte para a criação de regras em nível de aplicação na tabela de fluxo. Atualmente, as regras podem ser criadas somente com campos do pacote que estejam no nível 2 (Ethernet), no nível 3 (Rede) e/ou no nível 4 (Transporte) da camada OSI/Internet, sendo o foco principal nos níveis 2 e 3 [Jarraya et al. 2014].

Levando em consideração os fatores anteriormente citados, o presente documento visa investigar o suporte a regras no nível 5 nas tabelas de fluxo nos DE. Para tanto, será proposta uma extensão ao protocolo OpenFlow. Essa extensão visa possibilitar funcionalidades que utilizam regras baseadas na camada de aplicação (nível 5), permitindo, assim, soluções sem a necessidade de outras ferramentas, como, *middleboxes*. Conforme sugerido pelos engenheiros da Intel [Intel 2014], tal extensão poderia ser adicionada aos DEs e utilizada por aplicações que realizassem, por exemplo, inspeção do conteúdo de pacotes (Deep Packet Inspection - DPI).

Esta primeira parte do trabalho consiste no estudo do protocolo OpenFlow, no entendimento do problema, na proposta de implementação e em como serão feitos os experimentos e a avaliação. Na segunda parte (Trabalho de Graduação 2), será realizada a implementação e apresentados os resultados obtidos com os experimentos.

O restante deste trabalho está organizado como segue. A Seção 2 explica, de maneira sucinta, o funcionamento do protocolo e o que possibilitará a criação da extensão. Na Seção 3, será apresentada a proposta de extensão, a visão geral do projeto, os desafios pertinentes à implementação e como a avaliação será executada. A Seção 4 expõe os objetivos do Trabalho de Graduação 2, a metodologia e o cronograma de atividades a serem realizadas. Por fim, a Seção 5 conclui o documento.

2. OpenFlow

OpenFlow é o primeiro padrão de interface para comunicação entre a camada de controle e de encaminhamento de uma arquitetura SDN [ONF 2012b]. Ele descreve um protocolo aberto para permitir que aplicações programem as tabelas de fluxo dos dispositivos de encaminhamento. A arquitetura OpenFlow consiste de três elementos principais: dispositivos de encaminhamento compatíveis com OpenFlow, uma ou mais instâncias de controlador e o canal seguro para a comunicação entre os DEs e as instâncias de controlador [Lara et al. 2014], como visto na Figura 1.

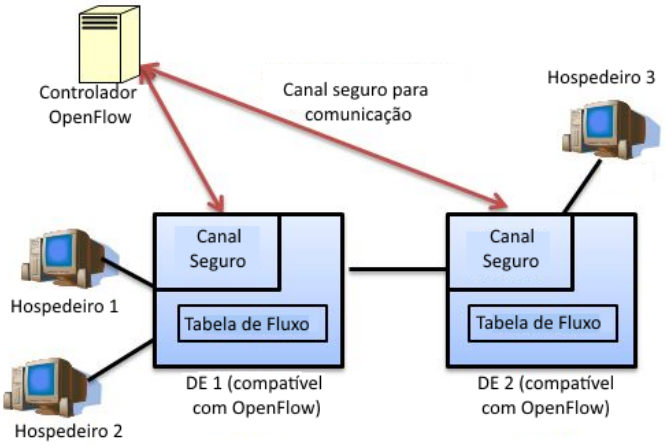

Figura 1. Arquitetura SDN e o protocolo OpenFlow. Baseado em [Lara et al. 2014]

Os DEs possuem uma tabela de fluxo (*Flow Table*) que armazena as regras de encaminhamento dos pacotes. Essa tabela consiste em uma lista de entradas de fluxo. Cada entrada de fluxo refere-se a uma regra e possui um conjunto de campos definidos de acordo com o protocolo OpenFlow [ONF 2013]. A Tabela 1 mostra a estrutura de uma entrada da tabela de fluxo, cujos campos são: *match*, prioridade, contadores, instruções, timeouts e cookie. Abaixo segue uma breve explicação sobre cada um desses campos.

- ***match:*** contém a regra para a realização do casamento.
- ***priority:*** define a precedência da entrada na tabela, afetando a ordem em que entradas serão processadas até haver o primeiro casamento.
- ***counters:*** contém estatísticas sobre a entrada de fluxo, sendo este campo atualizado quando um casamento ocorre.
- ***instructions:*** contém instruções que são aplicadas sobre o pacote após o casamento.
- ***timeouts:*** contém o máximo período de tempo que a entrada permanece válida no DE.
- ***cookie:*** valor escolhido pelo controlador para controle interno.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Tabela 1. Entrada de uma tabela de fluxo [ONF 2013]

O controlador, para definir o comportamento de encaminhamento dos dispositivos, deve, portanto, processar os campos dos pacotes que trafegam na rede e, a partir disso, programar as tabelas de fluxo dos dispositivos, informando as regras de encaminhamento.

Conforme citado anteriormente, ao que sabemos, não existe suporte a regras além do nível 4 do modelo OSI/Internet. Assim, quando um pacote entra no DE, ele passa por um processamento e é realizada a ação correspondente à entrada de maior prioridade na tabela. Caso não haja entrada que corresponda ao pacote, o DE deve ser programado para encaminhar o pacote para o controlador, que irá enviar uma nova entrada para o DE ou descartará o pacote. [1]

Na versão 1.2 do protocolo OpenFlow [ONF 2011], foi introduzida uma mudança nas mensagens que instalavam regras de fluxo nos dispositivos de encaminhamento. A mudança ocorreu na estrutura que especificava os campos de cabeçalho dos pacotes para as regras. Ela possibilitou que a extensão que este documento visa realizar fosse possível. Abaixo segue uma breve explicação sobre essa estrutura.

2.1. OpenFlow Extensible Match

A partir da versão 1.2 do protocolo, a estrutura utilizada para a instalação de regras no dispositivo de encaminhamento passou a ser da forma de um *Type-Length Value* (TLV), onde *type* é o tipo de dados, *length* o tamanho da carga útil e *value* a carga útil. O objetivo principal dessa mudança foi adicionar flexibilidade às mensagens que instalam as regras nos DEs. Essa flexibilidade favoreceu o suporte à introdução de novos protocolos ao protocolo OpenFlow. A essa estrutura dá-se o nome de *OpenFlow Extensible Match* (OXM).

Através da OXM, os campos de *matches* são definidos em mensagens e instalados no DEs. Devido à sua extensibilidade, ela possibilitou a criação de novos campos de *matches*. Além disso, viabilizou a criação de expressões flexíveis de *matches* nas mensagens que instalam as regras nos DEs, pois agora cada campo possui um tipo particular e é definido em um OXM específico – diferentemente da abordagem anterior, em que a mensagem era fixa e pré-definida.

Os campos são definidos na mensagem a partir da estrutura OXM, como mostra a Figura 2. Essa estrutura possui um cabeçalho de 32 bits, em que temos os seguintes campos:

- Class (16 bits): Agrupa os campos que serão usados para o *match*. Temos duas classes principais:
  - *OpenFlow Basic*: contém o conjunto básico de campos do pacote, definidos pelo protocolo OpenFlow (e.g.: protocolo, endereço MAC, . . .).
  - *Experimenter*: classe utilizada quando especificado algum *match* experimental.
- *Field* (7 bits): Varia conforme a definição da classe. Caso seja a classe *OpenFlow Basic*, irá conter os campos básicos do pacote que serão definidos para o *match* (e.g.: todos os campos que estão na classe OpenFlow Basic). [2]
- *HasMask* (1 bit): Define se existe campo "coringa" (máscara) na mensagem.
- *Length* (8 bits): Tamanho do *payload*.

Como visto na Figura 2, a mensagem para definição de regras no switch possui um vetor de OXMs, que pode ter zero (o que corresponde ao *match* coringa) ou mais OXMs.

[1] Mais informações sobre o funcionamento do mecanismo podem ser encontradas em [ONF 2013].

[2] Campos definidos na seção A.2.3.7 em [ONF 2011]

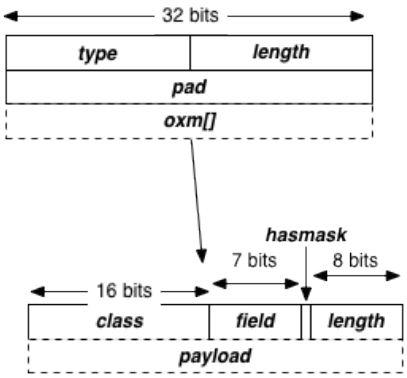

Figura 2. Estrutura de *Match* com OXM

Assim, quando o controlador define uma regra de *match*, a expressão da regra é formada a partir do conjunto dos OXMs presentes no vetor.

O presente documento propõe um novo campo de *match* com alcance ao nível de aplicação no pacote. A criação desse campo foi possível a partir dessa estrutura e da classe *EXPERIMENTER*.

2.2. Camada de Aplicação e OpenFlow

Apesar dos avanços que o OpenFlow possibilitou, ainda há muito o que se explorar para que seja possível usufruir da capacidade que as SDNs oferecem. Um assunto não muito explorado é o suporte em nível de aplicação no protocolo. Esse suporte, quando utilizado na rede, é chamado de *Deep Packet Inspection* (DPI), pois vai além dos cabeçalhos do pacote. Apesar da polêmica que existe em torno de DPI – é considerada uma tecnologia disruptiva, devido ao fato de entrar em aspectos de privacidade e equidade da rede [Bendrath and Mueller 2011] – seu uso poderia agregar benefícios para a rede, pois existem diversas aplicações que podem utilizar esse suporte e não estão relacionadas com tais aspectos.

Dentre essas aplicações, *Quality of Service* (QoS) e as relacionadas com segurança da rede privada, como *Firewall*, podem se valer de DPI, e seu uso, incorporado ao OpenFlow, traria benefícios e facilidades para as mesmas. Hoje, para a realização dessas funcionalidades, são necessários *middleboxes*, que acabam por adicionar latência e custo à rede. Segundo [Jarraya et al. 2014], funções de segurança na rede, como detecção de intrusão e de anomalia, necessitam de informações relacionadas ao pacote a diferentes níveis de detalhes e a diferentes taxas. Assim, o acesso ao *payload* é crucial para esse tipo de aplicação.

O *framework* AVANT-GUARD [Shin et al. 2013], que adiciona inteligência ao plano de dados para agregar segurança e resiliência ao protocolo OpenFlow, utiliza informações contidas no *payload* dos pacotes. Para contornar o problema da falta de acesso ao *payload*, foram criados *triggers* que o encaminham para o controlador, onde então é realizada a análise para detecção de intrusão. Eles verificaram que, apesar de contornar o problema, foi criado um gargalo entre o controlador e o dispositivo de encaminhamento.

Com o suporte ao processamento do *payload* das mensagens, regras de fluxo poderiam ser definidas em nível de aplicação e, assim, o tratamento de fluxos poderia ser

realizado com uma granularidade muito mais fina. Aplicações como *Quality of Service*, *Intrusion Detection System*, entre outras, poderiam ser realizadas sem a necessidade de *middleboxes*. Assim, apesar do papel importante que esse mecanismo desempenha para o desenvolvimento de aplicações de rede, ele foi pouco explorado até hoje, portanto serve de motivação para a realização do presente trabalho.

3. AppFlow: Visão Geral do Projeto

Dada a motivação e a importância desse suporte para o protocolo Openflow, o presente documento irá apresentar as questões relevantes ao projeto da extensão, nomeada *AppFlow*.

Diferentemente da abordagem utilizada em [Shin et al. 2013], o objetivo desse trabalho é adicionar o suporte em nível de aplicação aos dispositivos de encaminhamento. Dessa maneira a latência é reduzida e não criamos um *bottleneck* no canal de comunicação com o controlador. Para isso, criaremos um novo campo na tabela de fluxo dos DEs compatíveis com OpenFlow. Esse campo terá acesso ao *payload* do pacote e seu suporte será criado com o auxílio da classe *EXPERIMENTER* (Seção 2.1).

Porém, para a realização do projeto, será necessária a adaptação de alguns elementos da arquitetura OpenFlow, conforme Figura 3.

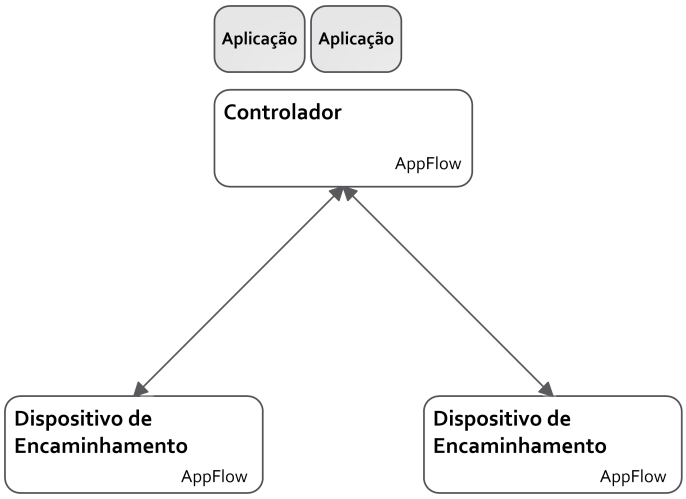

Figura 3. Visão geral do projeto *AppFlow*.

Segue a explicação sobre os elementos da Figura 3.

- **Dispositivo de encaminhamento com suporte *AppFlow***: será necessário desenvolver uma adaptação para o dispositivo de encaminhamento compreender e processar as regras em nível de aplicação, algo até então não fornecido nas implementações existentes, pois foge do escopo da definição do protocolo OpenFlow [ONF 2013].
- **Controlador com suporte *AppFlow***: na instância de controle da rede, será necessário o desenvolvimento de um módulo na arquitetura de controle, para que o controlador possa instalar as novas regras em nível de aplicação no DE.
- **Aplicação**: implementação de uma aplicação para validar a troca de mensagens entre o controlador e os DEs, bem como a inserção de regras em nível de aplicação nos DEs da rede.

De acordo com o explicitado, há dois pontos chave para a implementação da extensão: o controlador e o dispositivo de encaminhamento; ambos precisarão ter seus códigos alterados para suportar o *AppFlow*. A próxima seção evidencia esses quesitos e expõe as opções e dificuldades relacionadas à adaptação de cada componente.

3.1. Implementação

Controlador. Como requisito inicial, o controlador deve suportar a versão 1.2 do OpenFlow. Existem diversos controladores, alguns são citados por [Lara et al. 2014], porém a maioria deles só apenas oferece suporte à versão 1.0 do OpenFlow. Dentre estes, temos o controlador Ryu [Ryu 2014] e o controlador Floodlight-plus [SDN Hub 2014], ambos de código aberto e que suportam até a versão 1.3 do protocolo.

Entre esses dois, o que se destaca é o controlador Ryu. Por ser mais antigo, tem suporte à versão 1.3 do protocolo e algumas extensões propostas pela ONF [ONF 2014]. Portanto, esse controlador será utilizado como base para a implementação do AppFlow.

Para a implementação do código, o presente trabalho utilizará a estrutura OXM (Seção 2.1), em que, através do TLV, podemos criar um novo campo de *Match* do tipo *EXPERIMENTER*. Portanto, será implementado um módulo no código do controlador escolhido. Esse módulo deverá adicionar suporte ao novo campo e às mensagens pertinentes a ele, como a instalação, a modificação e a remoção de regras. Esse campo será definido com maiores detalhes no Trabalho de Graduação 2.

Dispositivo de Encaminhamento. DEs são classificados em *Switches Software-based* e *hardware-based OpenFlow-Enabled*. Alguns dos *switches* baseado em *software* são: Open vSwitch [OVS 2014], Pica8 [Pica8 2014], entre outros [3]. Implementações em hardware podem ser encontradas em diversas empresas, como HP, NEC, Pronto, Juniper, entre outras.

Como o AppFlow requer que o código do *Switch* seja modificado, precisamos de uma implementação em software. Este trabalho utilizará como base o *Open vSwitch* [OVS 2014], que possui código aberto e suporte à versão 1.3 do protocolo OpenFlow [ONF 2012a]. A partir dessa escolha, será implementado o código necessário para que as mensagens correspondentes às regras sejam reconhecidas e o código responsável por realizar o *match* (do novo campo em nível de aplicação) nas tabelas de fluxo.

A modificação do código do dispositivo de encaminhamento consistirá na parte mais complexa do trabalho. A complexidade não se deve apenas ao fato de que o *Open vSwitch* possui código de alta complexidade, mas também na dificuldade de realizar a busca no *payload* dos pacotes. Até o nível 4 da camada OSI/Internet, existe uma padronização entre os protocolos, de modo que podemos saber exatamente onde encontrar os valores dentro do pacote. No nível 5 (camada de aplicação), essa padronização não existe, cada aplicação pode escolher um maneira diferente para representar os dados e, possivelmente, teremos de optar por algum padrão de representação.

Desse modo, notamos que a implementação do AppFlow consiste em um fator de risco para o trabalho e irá exigir esforço e pesquisa para a realização dessa tarefa.

[3] Uma lista completa de *switches software-based* pode ser encontrado em [Jarraya et al. 2014]

3.2. Avaliação

Para a realização da avaliação, será utilizada uma topologia de rede com dispositivos de encaminhamento e controladores com suporte ao AppFlow. Isso será realizado através da ferramenta *Mininet* [Mininet 2014], que permite a criação de uma rede simulada com parâmetros realistas. Através da ferramenta, avaliaremos se o comportamento dos DEs e do controlador descrevem o esperado.

Para que o comportamento seja verificado adequadamente, será necessária a criação de uma aplicação que venha a utilizar regras com a camada de aplicação. A verificação poderia ser efetuada por um *Firewall* de Aplicações, um detector de intrusão ou um provedor de *Quality of Service* (QoS). A partir da aplicação, será possível realizar *benchmarks* e avaliar se o desempenho do AppFlow é satisfatório perante soluções já desenvolvidas e que realizam a mesma função (*middleboxes*). Uma possível métrica a ser utilizada é a latência adicionada ao pacote devido ao processamento do AppFlow. Essa medida tem como objetivo verificar se realmente é mais barato realizar tal processamento no *Switch*.

A partir destas avaliações, teremos parâmetros para determinar as vantagens, desvantagens e custos resultantes do uso dessa extensão.

4. Objetivos e Organização do Trabalho

Na segunda etapa deste Trabalho de Graduação, serão desenvolvidas as implementações necessárias e a avaliação da extensão AppFlow, conforme detalhado anteriormente. A partir destes resultados, será feita uma análise do que foi obtido com a extensão, bem como o impacto das mudanças necessárias sobre a organização da arquitetura SDN para a obtenção dessa funcionalidade. Como resultado, esperamos que a extensão tenha um tempo de processamento do pacote "razoável", comparado com *middleboxes* que realizam DPI, possibilitando, assim, uma redução nas despesas que essas soluções acarretam para as aplicações.

Essa seção apresenta a metodologia e o cronograma para a realização das tarefas pertinentes ao trabalho.

4.1. Metodologia

A seguinte metodologia está sendo empregada, visando os objetivos propostos.

1. **Estudo:** o objetivo principal é entender o problema e compreendê-lo. Para isso, esta etapa é composta de quatro tarefas: (i) estudo do paradigma SDN, com ênfase sobre o protocolo OpenFlow; (ii) estudo da estrutura OXM do protocolo OpenFlow e de aplicações que utilizem DPI; (iii) análise de arquiteturas de controladores e dispositivos de encaminhamento, suas características e funcionalidades; e (iv) levantamento bibliográfico do estado da arte, considerando a definição de regras em nível de aplicação para redes SDN.
2. **Análise:** avaliação da viabilidade do desenvolvimento de uma extensão, tomando como base o estudo realizado na etapa 1.
3. **Busca e escolha das ferramentas:** busca pelas ferramentas (arquiteturas de controladores e implementações de dispositivos de encaminhamento) que serão utilizadas como base para as implementações, levando em consideração as exigências e necessidades encontradas nas etapas 1 e 2.

4. **Implementação:** projeto e implementação do código do AppFlow no controlador e *switch* escolhidos.
5. **Avaliação:** avaliação da proposta, conforme descrito na Seção 3.2.

Até o presente momento, as etapas 1 e 2 foram realizadas e a etapa 3 está em fase final. Apesar disso, o estudo do problema e a análise continuarão durante o Trabalho de Graduação 2, a fim de superar possíveis dificuldades encontradas durante a implementação do *AppFlow*.

4.2. Cronograma

As tarefas a serem realizadas na segunda etapa do Trabalho de Graduação são enumeradas a seguir. A Tabela 2 apresenta o cronograma do trabalho.

1. Projeto do protótipo, descrevendo o modelo e ferramentas escolhidas.
2. Compreensão e estudo do código do *Open vSwitch* e do controlador Ryu.
3. Implementação do protótipo no controlador e no *switch*.
4. Avaliação dos resultados obtidos, conforme descrito na Seção 3.2.
5. Redação do Trabalho de Graduação 2.
6. Apresentação do Trabalho de Graduação 2.

Tarefa	**2014**						
	Jun	**Jul**	**Ago**	**Set**	**Out**	**Nov**	**Dez**
1	X						
2	X	X					
3		X	X	X	X		
4				X	X		
5					X	X	
6							X

Tabela 2. Cronograma de atividades para o Trabalho de Graduação 2

5. Considerações Finais

Através da realização desse trabalho, foi possível obter um melhor entendimento sobre *Software-Defined Networking*, o protocolo OpenFlow e as razões que levaram ao surgimento desses paradigmas. O protocolo OpenFlow proporcionou muitas vantagens para a área, porém ainda existem muitos aspectos a serem melhorados. Este trabalho pretende possibilitar ao protocolo OpenFlow evoluir em direção à relação de aplicações da camada de enlace com o processamento das mensagens no núcleo da rede. Por fim, através desse trabalho, foi possível melhor estruturar o problema para que o objetivo final seja atingido no Trabalho de Graduação.

Referências

Bendrath, R. and Mueller, M. (2011). The end of the net as we know it? deep packet inspection and internet governance. *New Media & Society*, 13(7):1142–1160.

Intel (2014). Service-aware network architecture based on sdn, nfv, and network intelligence. Disponível em <http://www.intel.com.br/content/dam/www/public/us/en/documents/white-papers/service-aware-network-architecture-sdn-nfv-paper.pdf>. Acesso em: 5 jun. 2014.

Jarraya, Y., Madi, T., and Debbabi, M. (2014). A survey and a layered taxonomy of software-defined networking. *Communications Surveys Tutorials, IEEE*, PP(99):1–1.

Lara, A., Kolasani, A., and Ramamurthy, B. (2014). Network innovation using openflow: A survey. *Communications Surveys Tutorials, IEEE*, 16(1):493–512.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

Mininet (2014). Mininet - An Instant Virtual Network on your Laptop (or other PC). Disponível em <http://mininet.org>. Acesso em: 6 jun. 2014.

Nunes, B., Mendonca, M., Nguyen, X., Obraczka, K., and Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys Tutorials, IEEE*, PP(99):1–18.

ONF (2011). Openflow switch specification version 1.2. Disponível em <http://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf>. Acesso em: 5 jun. 2014.

ONF (2012a). Openflow switch specification version 1.3.0. Disponível em <http://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf>. Acesso em: 12 jun. 2014.

ONF (2012b). Software-defined networking: The new norm for networks. ONF White Paper. Disponível em <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>. Acesso em: 5 jun. 2014.

ONF (2013). Openflow switch specification version 1.4.0. Disponível em <http://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>. Acesso em: 5 jun. 2014.

ONF (2014). Open Networking Foundation [Online]. Disponível em <http://www.opennetworking.org>. Acesso em: 4 jul. 2014.

OVS (2014). Open vSwitch - An Open Virtual Switch. Disponível em <http://openvswitch.org>. Acesso em: 5 jun. 2014.

Pica8 (2014). Pica8 Open Networking. Disponível em <http:/http://www.pica8.com/>. Acesso em: 12 jun. 2014.

Ryu (2014). Ryu SDN Framework [Online]. Disponível em <http://osrg.github.io/ryu/>. Acesso em: 6 jun. 2014.

SDN Hub (2014). Floodlight-Plus [Online]. Disponível em <http://bitbucket.org/sdnhub/floodlight-plus>. Acesso em: 6 jun. 2014.

Shin, S., Yegneswaran, V., Porras, P., and Gu, G. (2013). Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 413–424, New York, NY, USA. ACM.