UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

TRABALHO DE CONCLUSÃO DE CURSO

PROXY ROUTEFLOW BASEADO EM JAVA

FABIANO SILVA MATHILDE

ORIENTADOR: PROF. DR. CESAR AUGUSTO CAVALHEIRO MARCONDES
COORIENTADOR: DR. CHRISTIAN ESTEVE ROTHENBERG

São Carlos – SP

Dezembro/2012

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

TRABALHO DE CONCLUSÃO DE CURSO

PROXY ROUTEFLOW BASEADO EM JAVA

FABIANO SILVA MATHILDE

Dissertação apresentada ao Departamento de Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação, área de concentração: Sistemas Distribuídos e Redes

Orientador: Prof. Dr. Cesar Augusto Cavalheiro Marcondes

São Carlos – SP

Dezembro/2012

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

TRABALHO DE CONCLUSÃO DE CURSO

PROXY ROUTEFLOW BASEADO EM JAVA

FABIANO SILVA MATHILDE

Dissertação apresentada ao Departamento de Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação, área de concentração: Sistemas Distribuídos e Redes
Orientador: Prof. Dr. Cesar Augusto Cavalheiro Marcondes

Aprovado em 09 de Dezembro de 2012.

Membros da banca:

Prof. Dr. Cesar Augusto Cavalheiro Marcondes
(Orientador– DC-UFSCar)

São Carlos – SP

Dezembro/2012

RESUMO

Nos últimos anos o protocolo *OpenFlow* vem aumentando a visibilidade das tecnologias de redes definidas por software, fazendo com que um número cada vez maior de pesquisadores e desenvolvedores o adotem como principal ferramenta para simulações ou aplicações em ambientes reais. O projeto comunitário *RouteFlow* liderado pelo *CPqD* propõe uma plataforma de roteamento IP definido por software (do termo em inglês, *software-defined networking*) baseado no protocolo *OpenFlow*, que permite uma separação efetiva do plano de controle do plano de encaminhamento dos equipamentos de rede. O fato do sistema ter sido criado com o código totalmente aberto fez com que o número de usuários aumenta-se consideravelmente incentivando a equipe de desenvolvedores a atualizá-la constantemente para agregar cada vez mais ferramentas e tecnologias. O *RouteFlow* faz a manipulação do protocolo *OpenFlow* utilizando os softwares de controle mais famosos da literatura, o *NOX*, criado totalmente em *C++* e o *POX*, criado totalmente em *Python*. Para aumentar a capacidade do *RouteFlow* o trabalho em questão descreve a adição de suporte à um novo software de controle, o *Floodlight*, sendo criado totalmente em *Java*. Sendo assim o *RouteFlow* ganhará suporte a mais uma tecnologia mantendo-se sempre na vanguarda das tecnologias de redes definidas por software.

Palavras-chave: Redes Definidas por Software

ABSTRACT

Ainda a ser feito.

Keywords: Software Defined Network

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ISP – *Internet Service Provider*
IP – *Internet Protocol*
UDP – *User Datagram Protocol*
TCP – *Transmission Control Protocol*
OF – *OpenFlow*
ARP – *Address Resolution Protocol*
MPLS – *Multiprotocol Label Switching*
IPC – *Inter-Process Communication*
REST – *Representational State Transfer*
OSPF – *Open Shortest Path First*
BGP – *Border Gateway Protocol*
RIP – *Routing Information Protocol*
JSON – *JavaScript Object Notation*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Objetivo do Trabalho

O trabalho tem como objetivo principal agregar uma nova funcionalidade ao *RouteFlow*. A funcionalidade será o suporte nativo ao controlador *Floodlight*. Os controladores são usados pelo *RouteFlow* como uma interface de comunicação entre os softwares de roteamento e os switches *Openflow*. Cada controlador possui certas características juntamente com recursos exclusivos e com isso esperá-se que o *RouteFlow* agregue as melhores ferramentas disponíveis no controlador *Floodlight*. A implementação atual do RouteFlow possui suporte aos controladores *NOX* e *POX*, sendo desenvolvidos respectivamente em C++ e Python. O Floodlight foi totalmente desenvolvido em *Java* tendo suporte ao estilo de comunicação distribuída *REST (Representational State Transfer)*, sendo possível utilizado sem a necessidade de programação, através de mensagens *REST*.

1.2 Contribuições

Como principal contribuição do trabalho podemos citar a integração que haverá entre o *RouteFlow* e a comunidade de usuários do controlador *Floodlight*. A comunidade poderá realizar simulações ou até experimentos em ambientes reais contribuindo ainda mais com o avanço do *RouteFlow*. Outra contribuição importante que pode ser citada é a absorção pelo *RouteFlow* das melhores ferramentas providas pelo *Floodlight*, tornando-o cada vez mais completo.

Capítulo 2

REDES DEFINIDAS POR SOFTWARE

2.1 Definição Geral

As Redes Definidas por Software (Software Defined Networks, ou SDN) constituem um novo paradigma para o desenvolvimento de pesquisas em redes de computadores que vem ganhando a atenção de grande parte da comunidade acadêmica e da industria da área de redes. Fazendo um balanço geral da situação que encontramos hoje, podemos dizer que é um pouco complexa: é possível afirmar que a área de redes fez um sucesso estrondoso, já que hoje a tecnologia de redes de computadores permeia todos os níveis da sociedade. Grande parte das atividades da sociedade de alguma forma passa por uma ou mais redes de computadores.

Mas tamanho sucesso trouxe consigo um problema para a comunidade de pesquisa. Como grande parte da sociedade depende hoje da internet em suas atividades diárias e as tecnologias de rede se tornaram de fácil acesso, a estabilidade se tornou uma característica fundamental das redes de computadores. Isso significa que pesquisas com novas tecnologias e protocolos já não são mais possíveis em redes de larga escala, como a Internet, devido ao risco de interrupção ou instabilidade dos serviços essenciais. Outro problema encontrado pelos pesquisadores é o fato de que a larga utilização de tecnologias já desenvolvidas inviabiliza a inserção de qualquer tecnologia que exija a inserção de novos equipamentos de hardware.

Mesmo pesquisadores trabalhando em redes experimentais sofrem para justificar a adoção em larga escala das tecnologias desenvolvidas nesses ambientes. O potencial de instabilidade ou ruptura de tais avanços se torna um forte argumento contra sua adoção.

Esses problemas citados acima só ocorrem pelo fato de que redes de computadores em geral e a rede mundial (a Internet) atingiram um nível de amadurecimento que as tornaram pouco flexíveis. Para tentar melhorar essa situação, a comunidade de pesquisa em redes de

computadores tem investido em iniciativas que levam ao desenvolvimento de redes com maiores recursos de programação, de forma que as novas tecnologias possam ser inseridas na rede de forma gradual. Exemplos de iniciativas desse tipo são as propostas de redes ativas *(active networks)* [Tennenhouse e Wetherall 2007], de *testbeds* como o PlanetLab [Peterson e Roscoe 2006] e, mais recentemente, do GENI [Turner 2006, Elliott e Falk 2009]. Redes ativas, apesar de terem grande potencial, tiveram pouca aceitação pela necessidade de alteração dos elementos de rede para permitir que se tornassem programáveis. As iniciativas mais recentes, como o PlanetLab e o GENI, apostam na adoção de recursos de virtualização para facilitar a transição para novas tecnologias. Apenas de serem consideradas de grande potencial ao longo prazo, tais iniciativas ainda enfrentam desafios em questões como garantir o desempenho exigido pelas aplicações largamente utilizadas hoje utilizando-se tais elementos de rede virtualizados.

Uma outra forma de abordar o problema, a fim de oferecer um caminho de menor impacto e que possa ser implementado em prazos mais curtos e com bom desempenho, consiste em estender o hardware de encaminhamento de pacotes de forma mais restrita. Considerando-se que a operação que precisa de alto desempenho nos elementos de comutação atual é o encaminhamento de pacotes, algumas iniciativas propõem manter essa operação pouco alterada, para manter a viabilidade de desenvolvimento de hardware de alto desempenho, mas com uma possibilidade de maior controle por parte dos administradores de rede. Essa proposta se inspira em uma tecnologia já largamente adotada, o chaveamento (encaminhamento) baseado em rótulos programáveis, popularizado pelo MLPS *(Multi-protocol Label Switching)* [Davie e Farrel 2008, Kempf et al. 2001].

Com o MPLS, o controle fino sobre o tráfego de rede se torna possível ao se atribuir a cada pacote um rótulo *(label)* que determina como o mesmo será tratado pelos elementos de rede. Explorando esse recurso, administradores de rede podem exercer controle diferenciado sobre cada tipo de tráfego de rede, assumindo que os mesmos possam ser identificados para receberem os rótulos apropriados. Com base nessa observação, uma ideia trabalhada por diversos pesquisadores é a manutenção de um hardware de encaminhamento de alto desempenho, com a possibilidade de permitir que os administradores de redes (ou os desenvolvedores de aplicações para a rede) determinem como os fluxos irão ser rotulados e encaminhados.

A iniciativa mais bem sucedida nesse sentido foi, sem dúvida, definição da interface e do protocolo OpenFlow [McKeown et al. 2008]. Com o OpenFlow, os elementos de encaminhamento oferecem uma interface de programação simples que lhes permite estender o acesso e controle da tabela de consulta utilizada pelo hardware para determinar o próximo passo de cada pacote recebido. Dessa forma, o encaminhamento continua sendo eficiente, pois a consulta à

tabela de encaminhamento continua sendo tarefa do hardware, mas a decisão sobre como cada pacote deve ser processado pode ser transferida para um nível superior, onde diferentes funcionalidades podem ser implementadas. Essa estrutura permite que a rede seja controlada de forma extensível através de aplicações, expressas em software. A esse novo paradigma, deu-se o nome de Redes Definidas por Software, ou *Software Defined Networks* (SDN).

Do ponto de vista histórico, as Redes Definidas por Software têm sua origem na definição da arquitetura de redes *Ethane*, que definia uma forma de se implementar políticas de controle de acesso de forma distribuída, a partir de um mecanismo de supervisão centralizado [Casado et al. 2009]. Naquela arquitetura, cada elemento de rede deveria consultar o elemento supervisor ao identificar um novo fluxo. O supervisor consultaria um grupo de políticas globais para decidir, com base nas características de cada fluxo, como o elemento de encaminhamento deveria tratá-lo. Essa decisão seria comunicada ao comutador na forma de programação de uma entrada em sua tabela de encaminhamento com uma regra adequada para o novo fluxo (que poderia, inclusive, ser seu descarte). Esse modelo foi posteriormente formalizado por alguns autores na forma da arquitetura *OpenFlow*.

2.2 Introdução ao Protocolo *OpenFlow*

O *OpenFlow* foi proposto pela Universidade de Stanford para atender à demanda de validação de novas propostas de arquiteturas e protocolos de rede (incluindo as abordagens *clean slate*) sobre equipamentos comerciais. É definido como uma padrão aberto para Redes Definidas por Software que tem como principal objetivo que se utilize equipamentos de redes comerciais para pesquisa e experimentação de novos protocolos de rede, em paralelo com a operação normal das redes. Isso é conseguido com a definição de uma interface de programação que permite ao desenvolvedor controlar diretamente os elementos de encaminhamento de pacotes presentes no dispositivo. Com o *OpenFlow*, pesquisadores podem utilizar equipamentos de rede comerciais, que normalmente possuem maior poder de processamento que os comutadores utilizados em laboratórios de pesquisa, para realizar experimentos em redes ”de produção”. Isso facilita muito a transferência dos resultados de pesquisa para a industria.

2.3 Descrição Geral Protocolo *OpenFlow*

Uma característica básica da arquitetura do padrão *OpenFlow* é a separação clara entre os planos de dados e controle em elementos de chaveamento. O plano de dados cuida do

encaminhamento de pacotes com base em regras simples (chamadas de ações na terminologia *OpenFlow*) associadas a cada entrada da tabela de encaminhamento do comutador de pacotes (podendo ser um *switch*, um roteador ou até mesmo um ponto de acesso sem fio). Essas regras, definidas pelo padrão, incluem (*i*) encaminhar o pacote para uma porta especifica do dispositivo, (*ii*) alterar parte de seus cabeçalhos, (*iii*) descartá-los, ou (*iv*) encaminhá-lo para inspeção por um controlador da rede. Em dispositivos dedicados, o plano de dados pode ser implementado em hardware utilizando os elementos comuns aos roteadores e *switches* atuais. Já o módulo de controle (ou controladoor) pode ser um módulo de software implementado de forma independente em algum ponto da rede.

A principal abstração utilizada na especificação *OpenFlow* é o conceito de fluxo. Um fluxo é constituído pela combinação de campos do cabeçalho do pacote a ser processado pelo dispositivo, conforme Figura 2.1. As tuplas podem ser formadas por campos das camadas de enlace, de rede ou de transporte, segundo o modelo *TCP/IP*. Deve-se enfatizar que a abstração da tabela de fluxos ainda está sujeita a refinamentos, com o objetivo de oferecer uma melhor exposição dos recursos do hardware e, nesse caso, permitir a concatenação de várias tabelas já disponíveis, como, por exemplo, tabelas *IP/Ethernet/MPLS*. Nesse sentido, a contribuição mais importante do paradigma do *OpenFlow* é a generalização do plano de dados – qualquer modelo de encaminhamento de dados baseado na tomada de decisão fundamentada em algum valor, ou combinação de valores, dos campos de cabeçalho dos pacotes pode ser suportado.

inport	Ethernet			VLAN		IP				TCP/UDP	
	src	dst	type	id	pri	src	dst	proto	ToS	src	dst

Figura 2.1: Cabeçalho *OpenFlow* para a especificação dos fluxos

Um grande trunfo da arquitetura *OpenFlow* é a flexibilidade que ela oferece para se programar de forma independente do tratamento de cada fluxo observado, do ponto de vista de como o mesmo deve (ou não) ser encaminhado pela rede. Basicamente, o padrão *OpenFlow* determina como um fluxo pode ser definido, as ações que podem ser realizadas para cada pacote pertencente a um fluxo e o protocolo de comunicação entre o comutador de pacotes e o controlador, utilizado para realizar alterações dessas definições e ações. A união de uma definição de fluxo e um conjunto de ações forma uma entrada da tabela de fluxos *OpenFlow* [McKeown et al. 2008].

Em um *switch OpenFlow*, cada entrada na tabela de fluxos pode ser implementada como um padrão de bits representado em uma memória TCAM (Ternary Content- Addressable Memory). Nesse tipo de memória, bits podem ser representados como zero, um ou ”não

importa"(*don't care*), indicando que ambos os valores são aceitáveis naquela posição. Como o padrão é programado a partir do plano de controle, fluxos podem ser definidos da forma escolhida pelo controlador. A figura 2.2 apresenta uma visão geral de uma entrada da tabela *OpenFlow*. Cada pacote que chega a um comutador *OpenFlow* é comparado com cada entrada dessa tabela; caso um casamento seja encontrado, considera-se que o pacote pertence àquele fluxo e aplica-se as ações relacionadas à esse fluxo. Caso um casamento não seja encontrado, o pacote é encaminhado para o controlador para ser processado – o que pode resultar na criação de uma nova entrada para aquele fluxo. Além das ações, a arquitetura prevê a manutenção de três contadores por fluxo: pacotes, *bytes* trafegados e duração do fluxo. Esses contadores são implementados para cada entrada da tabela de fluxos e podem ser acessados pelo controlador através do protocolo *OpenFlow*.

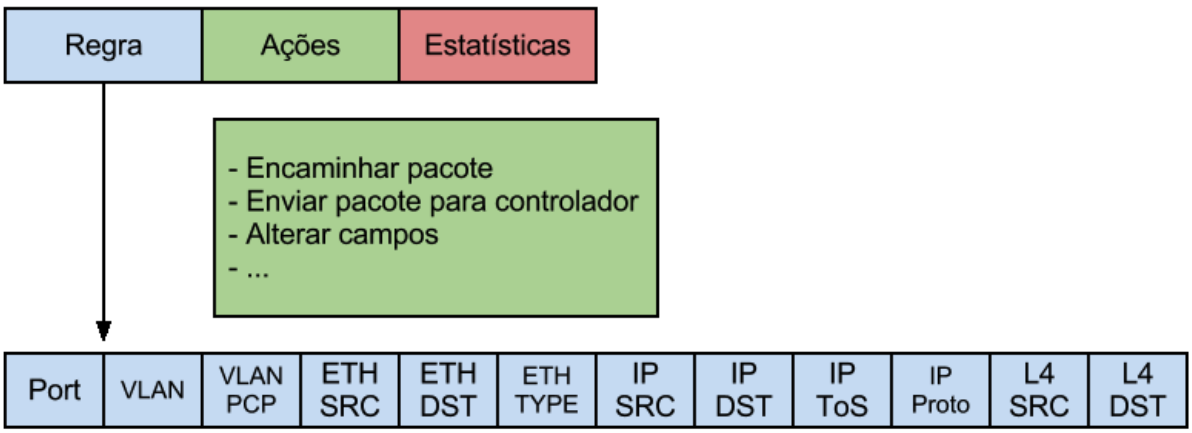

Figura 2.2: Exemplo de uma entrada na tabela de fluxos *OpenFlow*.

Esse pequeno conjunto de regras cria diversas possibilidades, pois muitas das funcionalidades que são implementadas separadamente podem ser agrupadas em um único controlador *OpenFlow*, utilizando um pequeno conjunto de regras. Alguns exemplos das possibilidades são apresentadas na Figura 2.3. As entradas representam o uso do *switch OpenFlow* para realizar encaminhamento de pacotes na camada de enlace, implementar um firewall e realizar encaminhamento de pacotes na camada de enlace utilizando redes virtuais (VLANs), respectivamente.

Apesar de possuir um conjunto pequeno de ações simples, alguns descrevem o *OpenFlow* como uma analogia ao conjunto de instruções de um microprocessador x86 que, apesar de pequeno e simples, provê uma vasta gama de possibilidades para o desenvolvimento de aplicações. O *OpenFlow* cria possibilidades semelhantes para o desenvolvimento de aplicações no contexto de redes de computadores.

A versão atual do *OpenFlow* ainda possui algumas limitações em termos do uso padrão em circuitos ópticos e uma definição de fluxos que englobe protocolos que não fazem parte do

Port	VLAN	ETH SRC	ETH DST	IP SRC	IP DST	IP Proto	L4 SRC	L4 DST	Ações
*	*	*	00:4F:...	*	*	*	*	*	**Porto 4**
*	*	*	*	*	*	**TCP**	*	**22**	**DROP**
*	**1**	*	00:4F:...	*	*	*	*	*	**Porto 4, 6, 9**

Figura 2.3: Exemplos de uso de um *Switch OpenFlow*.

modelo TCP/IP. No entanto, está sendo formulada uma nova versão cujo objetivo é eliminar algumas dessas limitações.

A Figura 2.4 define de forma introdutória uma rede de computadores com o protocolo *OpenFlow* habilitado. Os elementos comutadores podem ser de qualquer tipo, como comutadores convencionais, roteadores ou até mesmo pontos de acesso sem fio. Podemos ver o elemento externo, chamado de controlador, tomando contas das regras e ações instaladas no hardware de rede. O controlador pode ser executado em qualquer equipamento com suporte à redes, como um servidor comum.

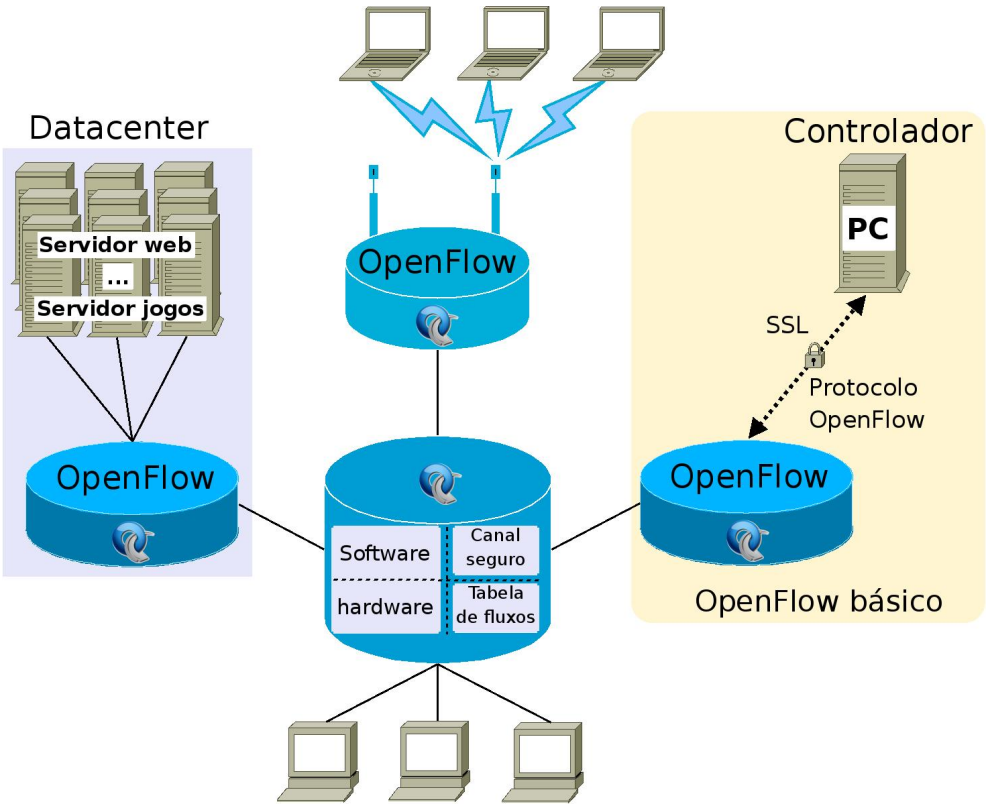

Figura 2.4: Rede com o protocolo *OpenFlow* Habilitado.

Capítulo 3

ARQUITETURA BÁSICA DO ROUTEFLOW

3.1 Introdução ao Projeto Comunitário RouteFlow

O projeto comunitário RouteFlow é uma proposta de oferta de serviços de roteamento IP remoto de forma centralizada, e que visa um desacoplamento efetivo entre o plano de encaminhamento e o plano de controle (ROUTEFLOW, 2011). O objetivo é tornar as redes IP mais flexíveis pela facilidade de adição, remoção e especialização de protocolos e algoritmos.

O RouteFlow prove um serviço virtualizado de roteamento IP em equipamentos com suporte ao protocolo *OpenFlow* seguindo o paradigma das redes definidas por software. Basicamente, o RouteFlow interliga uma infraestrutura *OpenFlow* com um ambiente virtual de roteamento IP baseado em ferramentas nativas do Linux (Quagga) para um roteamento eficiente na estrutura física. Baseado em um sistema de controle RouteFlow, os *switches* são instruídos via controladores *OpenFlow* que trabalham como proxies no intuito de traduzir as mensagens e eventos entre o ambiente físico e virtual.

O projeto conta com um número crescente de usuários no mundo (cerca de 1000 downloads e mais de 10000 visitantes desde que o projeto começou em Abril de 2010). As principais formas de contribuição para o projeto é através do sistema de repositórios GitHub. Para citar alguns exemplos de contribuição por parte da comunidade, é possível citar a contribuição do Google nos plugins SNMP e o trabalho atual no suporte ao MPLS e nas APIs de roteamento Quagga. A universidade americana de Indiana contribuiu com uma interface gráfica e com a execução de um piloto em seu *testbed*.

O RouteFlow é composto basicamente por três módulos principais: o cliente RouteFlow (RFClient), o servidor RouteFlow (RFServer) e o proxy RouteFlow. A Figura 3.1 mostra de forma simplificada um cenário RouteFlow típico: engines de roteamento em um ambiente

virtual geram uma base de informação de encaminhamento baseado nos protocolos de roteamento (OSPF, BGP) e processos ARP. Durante a execução do ambiente virtual, as tabelas de roteamento e tabelas ARP são coletadas pelos módulos RFClient executando nas máquinas virtuais e traduzido em tuplas OpenFlow que são enviadas para o RFServer, que faz a adaptação das informações para o ambiente físico e finalmente instrui o módulo RFProxy, uma aplicação controladora, para configurar os *switches OpenFlow* no ambiente físico.

Pacotes de encaminhamento dos protocolos de roteamento e controle de trágico (ARP, BGP, RIP e OSPF) são direcionados pelo RFProxy para as interfaces virtuais correspondentes do ambiente virtual. Entre o ambiente virtual e o ambiente existe um switch virtual que também é controlado pelo RFProxy, permitindo um caminho direto entre os ambientes, reduzindo o tempo das trocas de mensagens e sem a necessidade de passar através do RFServer e do RFClient.

Abaixo temos os principais avanços da arquitetura do projeto RouteFlow

- **Plataforma totalmente modular, extensível, configurável e flexível.** A arquitetura atual do RouteFlow foi feita pensando em sua própria evolução. Através do conceito de camadas, o RouteFlow conseguiu facilitar o entendimento geral do código, facilitando o seu entendimento total por parte de seus usuários. A construção baseada em módulos independentes torna o código mais claro, facilitando a portabilidade para outras linguagens ou o suporte à novas tecnologias. O exemplo mais claro sobre a independência dos módulos é o próprio módulo levado em consideração na construção do trabalho de conclusão de curso em questão, o RFProxy.

  O RFProxy faz a troca de mensagens com os outros módulos através de um sistema de banco de dados, sendo facilmente portado para outras linguagens de programação e tecnologias.

  Outro beneficio da construção baseada em módulos é a possibilidade de execução do RouteFlow em múltiplos sistemas computacionais, fazendo-o executar como uma arquitetura distribuída. Tal característica será melhor explica nos itens abaixo.

- **Suporte à replicação do estado da rede e grande disponibilidade dos recursos.** Para melhorar a disponibilidade do sistema, o RouteFlow foi construído de forma descentralizada, separando os dados relacionados ao estado das redes dos módulos de processamento. Todos os dados relacionados ao estado das redes é armazenado em um banco de dados centralizado, possibilitando que qualquer aplicação registrada tenha acesso. Dessa maneira é possível obter uma replicação dos processos, obtendo as vantagens e benefícios

de um sistema descentralizado.

A versão atual do projeto ainda não faz duplicação dos processos mas a ideia já é levada em consideração para futuras implementações.

- **Armazenamento do histórico da rede e de estatísticas.** Como citado no item anterior, a ideia de descentralização do RouteFlow fez que os dados fossem armazenados em um banco de dados centralizado. Esse características traz consigo uma série de vantagens, além de todas citadas anteriormente, ainda pode ser mencionado a possibilidade de se manter um histórico das ações e decisões tomadas pelo sistema. O RouteFlow, através do banco de dados centralizado, mantem um histórico de todo o sistema, bem como as estatísticas relacionadas à criação de regras e ao uso da rede. Tais características permitem aos pesquisadores ou até mesmo aos administradores de redes ter um controle e um entendimento mais elaborado, podendo reproduzir o ambiente em certos períodos de tempo. Sendo possível reproduzir o sistema em ambientes em que o mesmo apresentou certa irregularidade ou falha. As estatística ainda dão ao administrador um entendimento maior de como a rede é usada, possibilitando ao mesmo a tomada de alguma decisão para a busca de melhorias.

- **Possibilidade de ambientes com múltiplos controladores *OpenFlow*.** A arquitetura atual do RouteFlow foi desenvolvida para ter suporte em cenários com múltiplos controladores *OpenFlow*. Tal característica permite aos pesquisadores ou administradores particionar a rede e controlar cada região com um controlador independente. Outro fator interessante é que as camadas superiores do RouteFlow abstraem as diferenças entre as versões 1.0/1.1/1.2/1.2 do OpenFlow, tornando fácil o suporte à controladores heterogêneos. Para que tal característica fosse suportada, o código do RouteFlow passou por um processo de padronização, melhorando ainda mais a legibilidade do projeto.

3.2 Banco de Dados Centralizado com Suporte a IPC

Nesse capitulo será explicado com detalhes a arquitetura de banco de dados centralizado e seu suporte aos mecanismos de comunicação entre processos usados pelo projeto RouteFlow. Também serão abordados as principais vantagens e desvantagens da arquitetura.

Várias abordagens foram proposta pelo RouteFlow para um esquema unificado de comunicação entre processos (IPC). Inúmeras delas foram testadas e avaliadas até se conseguir traçar as principais vantagens e desvantagens de cada uma delas. Soluções baseadas em filas de mensagens, como o RabbitMQ ou o ZeroMQ foram descartadas por causa da grande

complexidade de implementação e manutenção. Soluções baseadas em serialização de mensagens, como ProtoBuffers e Thrift, se apresentaram como boas soluções mas requiriam uma lógica adicional para o armazenamento pendente e para mensagens já consumidas. Durante o estudo de banco de dados Não SQL para armazenamento persistente, surgiu as primeiras ideias do uso de um banco de dados como ponto central de um mecanismo de troca de mensagens entre processos (IPC) e consequentemente manter o histórico das ações tomadas pelo RouteFlow para permitir a replicação de certas situações.

Depois de levar em consideração as mais populares opções de banco de dados Não SQL (MongoDB, Redis, CouchDB), foi decidido sobre a implementação de um banco de dados centralizado e dos mecanismos de troca de mensagens entre processos (IPC) utilizando-se o MongoDB. Os principais fatores para a escolha foram a facilidade de programação, suporte nativo a inúmeras linguagens de programação, suporte nativo à tecnologia JSON e a existência de mecanismo para replicação e distribuição. A ideia por trás do mecanismo de troca de mensagens (IPC) é completamente independente da escolha do banco de dados e por isso não foi levada em consideração.

No núcleo do RouteFlow estão os mapeamentos entre o ambiente físico controlado e o ambiente virtual executando as tarefas de roteamento. A confiabilidade desses estados de rede é imprescindível para o RFServer e se torna difícil de manter sem a ajuda de algum módulo externo. Um banco de dados externo se encarrega desse objetivo, tendo sua configuração mais flexível. Estatísticas coletadas pelo RFProxy também podem ser armazenadas em um banco de dados centralizado, baseado em serviços adicionais é até possível implementar ferramentas para análise dos dados ou até mesmo visualização.

A escolha de delegar o controle dos estados da rede para um banco de dados permite uma melhor tolerância a falhas, replicando a base de dados ou até mesmo separando o RFServer em várias instâncias. A possibilidade de distribuição do RFServer permite ao sistema um melhor desempenho, sendo possível distribuí-lo em vários pontos de uma rede e assim reduzindo a latência de comunicação. Já é levado em consideração o uso de um banco de dados distribuído para as próximas atualizações do RouteFlow.

O RouteFlow armazena a lógica de controle dos *switches OpenFlow* na infraestrutura de rede, através de uma rede virtual composta por máquinas virtuais, cada uma executando um código (engine) de roteamento de domínio público (open source). Essas máquinas virtuais podem ser interconectadas de maneira a formar uma topologia lógica, espelhando a topologia de uma rede física correspondente ou uma topologia virtual simplificada. O ambiente virtual é armazenado em um servidor externo, ou um conjunto deles, que se comunicam com os equi-

pamentos do plano de dados através de um controlador *OpenFlow*, que transporta para o plano de encaminhamento as decisões tomadas pelos protocolos de roteamento no plano de controle (OSPF, BGP, RIP).

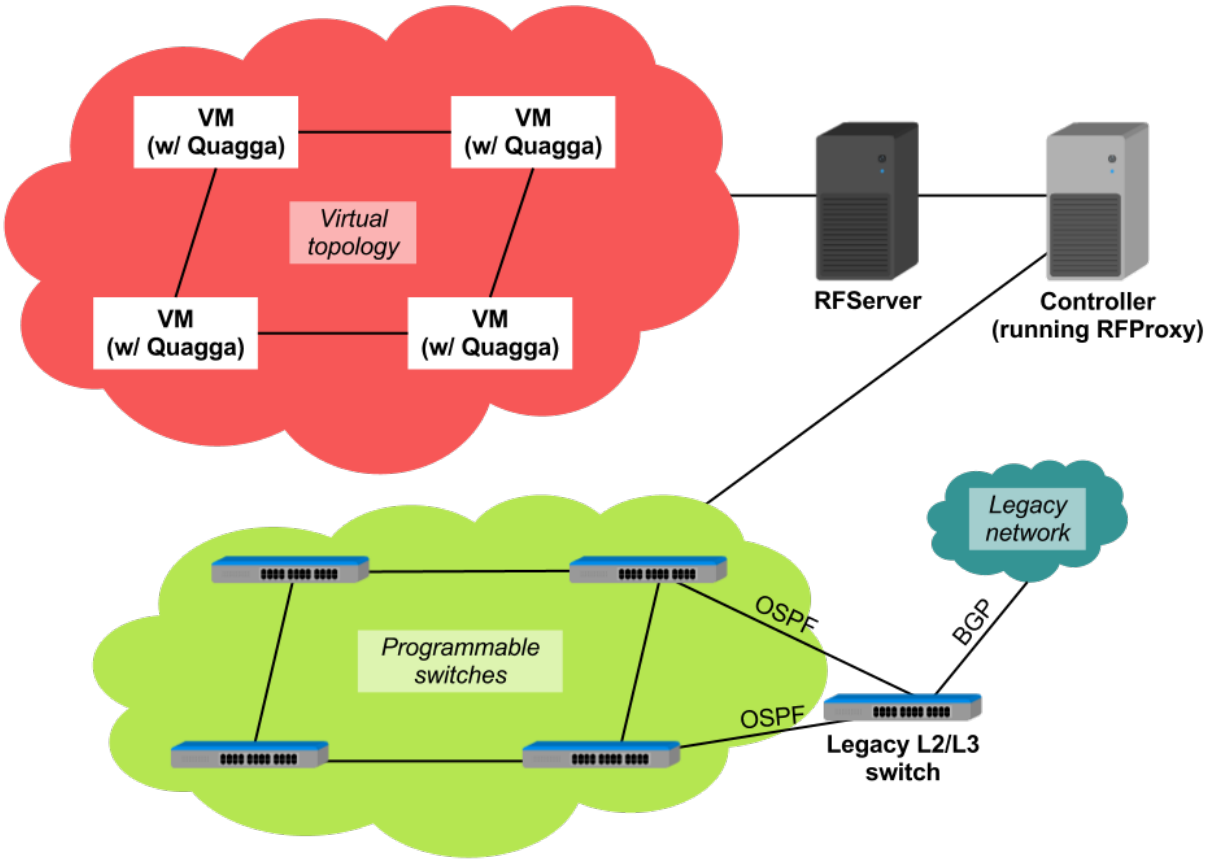

Figura 3.1: Visão geral do RouteFlow.

A Figura 3.1 ilustra uma sub-rede com switches programáveis, em que a lógica de roteamento é implementada no servidor RouteFlow. O resultado consiste numa solução flexível de alto desempenho e comercialmente competitiva, a partir da combinação de recursos disponíveis, como, por exemplo:

a) switches programáveis de baixo custo e software embarcado reduzido (*OpenFlow*);

b) pilhas de protocolos de roteamento open source (QUAGGA, 2009; XORP, 2011); e

c) servidor de prateleira de alto poder de processamento e, também, de baixo custo.

Cabe ressaltar que, apesar de o controle estar fisicamente centralizado, ele continua distribuído logicamente. Dessa forma, não é necessária qualquer alteração dos protocolos de roteamento existentes. Além disso, a solução pode tornar-se mais escalável no futuro, com o uso de vários servidores de alto desempenho.

3.3 Descrições dos Principais Componentes do RouteFlow

O RouteFlow é dividido basicamente em três aplicações básicas: RFClient, RFServer e RFProxy. Na Figura 3.2 temos uma visão geral das aplicações:

- RFClient executa como um programa executável em um máquina virtual, detectando mudanças na tabela ARP do Linux e na tabela de roteamento. As informações de roteamento são enviadas para o RFServer quando são atualizadas.

- RFServer é uma aplicação independente que gerencia as máquinas virtuais que estão executando o RFClient.

  O RFServer mantem o mapeamento entre as instâncias das máquinas virtuais executando o RFClient e as interfaces correspondentes aos switches e suas respectivas portas. É conectado ao RFProxy para instruí-lo como configurar os fluxos e também como configurar o Open vSwitch que mantém a conectividade de todo o ambiente composto pelas máquinas virtuais.

- RFProxy é uma aplicação responsável pelas interações entre os switches *OpenFlow* (identificados pelos seus datapaths) via o protocolo *OpenFlow*. Ele aguarda instruções do RFServer e o notifica à respeito de eventos na rede. Atualmente é executado como um módulo vinculado aos controladores *OpenFlow*.

  O RouteFlow tem suporte aos controladores NOX e POX, sendo que a proposta do trabalho é adicionar suporte ao Floodlight.

3.4 Protocolo RouteFlow

É o protocolo desenvolvido e usado para a comunicação entre os componentes do RouteFlow. Nele, estão definidas as mensagens e os comandos básicos para conexão e configuração das máquinas virtuais e, também, gerenciamento das entradas de roteamento em hardware. Entre os campos da mensagem-padrão estão: identificação do controlador, identificação da máquina virtual, tipo da mensagem, comprimento e dados.

O proxy RouteFlow recebe os comandos do servidor RouteFlow através deste protocolo e de acordo com o tipo de comando executa as principais ações, muitas delas exigindo a comunicação com os switches físicos. A comunicação com os switches físicos é feita via protocolo *OpenFlow*, fazendo o proxy RouteFlow agir como uma espécie de ”tradutor”entre os dois protocolos.

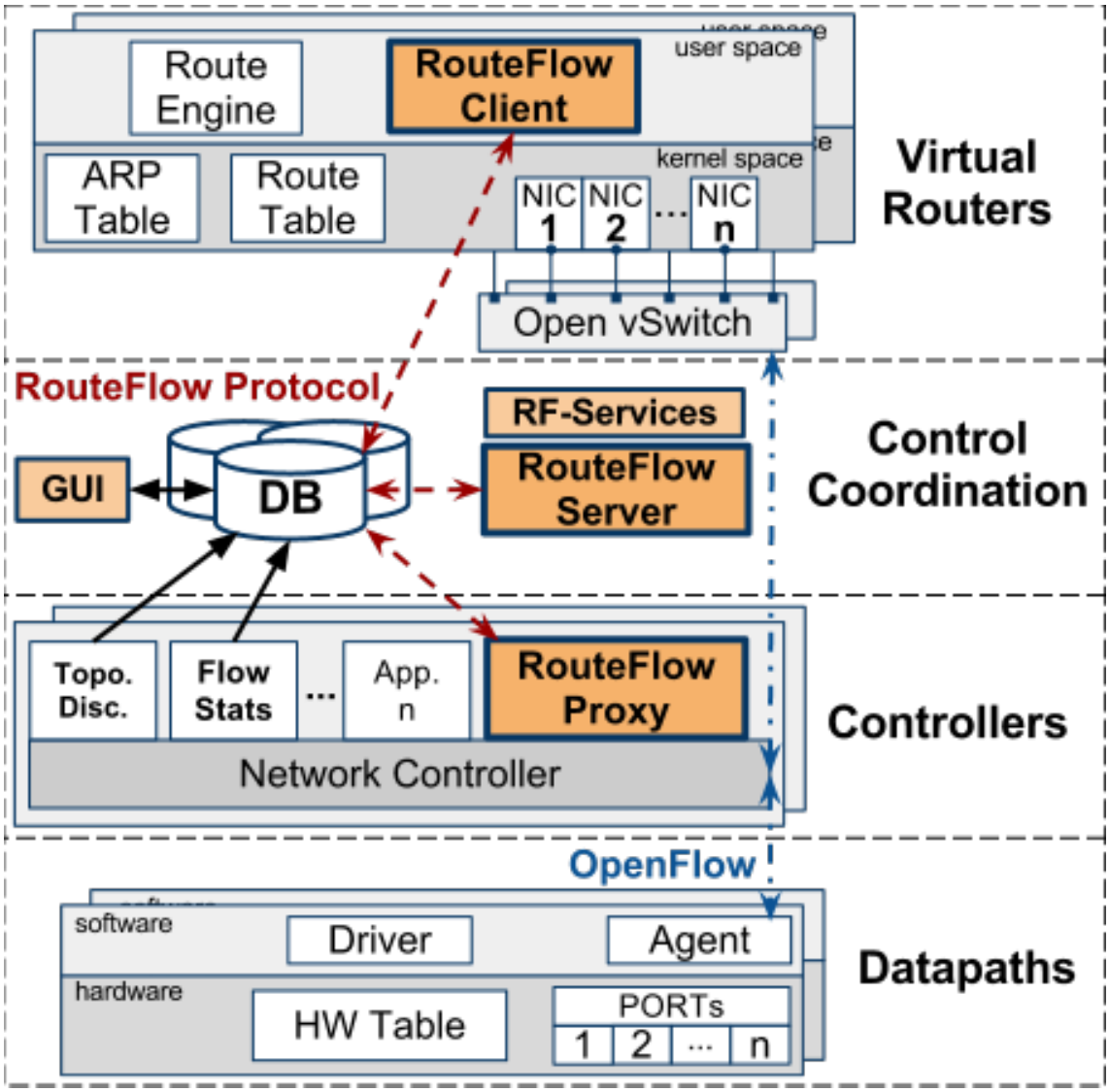

Figura 3.2: Componentes principais do RouteFlow.

Capítulo 4

PROXY ROUTEFLOW EM JAVA

4.1 Descrição Geral da Estrutura do Proxy RouteFlow em Java

Todos os componentes do proxy RouteFlow em Java foram agrupados em classes. O agrupamento em classes facilita a organização geral do código bem como a sua futura manutenção. Abaixo temos os componentes do proxy com suas respecitivas descrições:

- *MongoIPCMessageService:* Responsável pela comunicação entre o servidor RouteFlow e o proxy RouteFlow. A arquitetura básica do projeto RouteFlow faz uso de um banco de dados não SQL para troca de mensagens entre seus componentes, sendo que o banco de dados escolhido foi o MongoDB.

  O MongoDB possui alto desempenho sendo totalmente escrito em C++, outro aspecto importante é o fato de não ser SQL, o que facilita a sua integração com as principais linguagens de programação. O RouteFlow cria inúmeras tabelas no banco de dados, cada uma responsável pela comunicação entre um par de componentes, como toda comunicação é feita através de um sistema de banco de dados é possível que a comunicação entre componentes seja feita de forma simples, sem nenhum vinculo com a linguagem de implementação do mesmo.

  O componente MongoIPCMessageService cria um IPC (Inter-Process Communication) entre o servidor RouteFlow e o proxy RouteFlow. Os comandos são enviados de um componente para o outro na forma de mensagens pré-definidas. Cada mensagem define uma ação à ser tomada em relação aos eventos que vão ocorrendo ao longo da execução do ambiente. No corpo da mensagem estão os parâmetros que deverão ser usados para tomada da ação. Todas as mensagens que são colocadas na tabela pelo servidor Route-

Flow possuem um campo que indica se a mesma já foi tratada e em caso negativo cabe ao proxy tomar a ação e atualizar o campo da mensagem. Para tratamento das mensagens é gerado uma thread em looping infinito cujo único proposito de existência é o tratamento de novas mensagens. Essa característica pretende ser melhorada nas próximas versões do RouteFlow;

- *RFProtocolFactory:* Responsável pela criação das mensagens do protocolo RouteFlow. Cada tipo de mensagem RouteFlow é representada por um código e por uma respectiva classe. É papel do RFProtocolFactory retornar os objetos de mensagens à partir de seu código. Esta estrutura facilita a inserção de novas mensagens no ambiente evitando a reprogramação de outras classes;

- *RFProtocolProcessor:* Responsável pelo processamento de mensagens vindas do servidor RouteFlow. Este componente define o como será tratada cada mensagem vinda do servidor RouteFlow. As mensagens são lidas através do IPC e repassadas para tratamento.

- *AssociationTable:* Tabela mantida pelo proxy para armazenar a associação entre portas no ambiente virtual e porta no ambiente físico.

4.2 Descrição das Mensagens Traduzidas pelo Proxy RouteFlow

- *PortRegister*

- *PortConfig*

- *DatapathConfig*

- *RouteInfo*

- *FlowMod:* Mensagem utilizada para solicitação da instalação de regras nos switches OpenFlow. As mensagens FlowMod possuem em seu corpo um conjunto de parâmetros que define uma nova regra a ser aplicada à um switch OpenFlow. Cabe ao proxy criar a regra e envia-la corretamente ao switch. Para envio das regras é necessário a manipulação do protocolo OpenFlow. O Floodlight fornece uma série de funções para esse proposito, sendo usado como uma API de comunicação entre os switches OpenFlow e o proxy.

- *DatapathPortRegister:* Mensagem utilizada para registrar as portas dos switches OpenFlow no servidor RouteFlow. Esse registro é feito para que cada porta seja associada à uma porta da máquina virtual do ambiente virtual.

- *DatapathDown:* Mensagem utilizada para que o proxy informe ao servidor RouteFlow sobre a desconexão de um switch OpenFlow. O Floodlight, no papel de controlador OpenFlow, mantem um conjunto de informações à respeito dos switches OpenFlow ativos, podendo detectar quedas nas conexões dos mesmos. O servidor RouteFlow necessita ter esse tipo de informação para possíveis alterações nas regras dos switches OpenFlow.

- *VirtualPlaneMap:* Mensagem utilizada para que o proxy associa cada porta do switch OpenFlow à uma porta de uma máquina virtual no ambiente virtual.

- *DataPlaneMap:* Mensagem utilizada para que o servidor RouteFlow informe ao proxy à respeito de uma associação de porta bem sucedida. O proxy mantém uma tabela de associação entre portas no ambiente virtual e portas no ambiente físico.

Capítulo 5

RESULTADOS

5.1 Resultados Preliminares

Como resultado inicial pode ser citado o suporte

Capítulo 6

CONCLUSÃO

6.1 Trabalhos Futuros

Capítulo 7

AGRADECIMENTOS

- Allan Vidal – CPqD, Campinas - SP
- Cesar Augusto Cavalheiro Marcondes – UFSCar, São Carlos - SP
- Christian Esteve Rothenberg – CPqD, Campinas - SP
- Eder Leão Fernandes – CPqD, Campinas - SP
- Jorge Henrique de Barros Assumpção – CPqD, Campinas - SP
- Toda a equipe do Forum do Floodlight

Capítulo 8

REFERENCIAS