**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

Janeiro/2013

**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

Janeiro/2013

# 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 28 de Janeiro de 2013.

Membros da banca:

---

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

---

Prof. Dr. Hermes Senger
(DC - UFSCar)

---

Prof. Dr. Hélio Crestana Guardia
(DC - UFSCar)

São Carlos – SP

Janeiro/2013

# 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 pela Fundação *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 consideravelmente, incentivando a equipe de desenvolvedores a atualizá-la constantemente para agregar cada vez mais ferramentas e tecnologias. *RouteFlow* faz a manipulação do protocolo *OpenFlow* utilizando os softwares de controle mais populares 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 a um novo software de controle, o *Floodlight*, criado totalmente em *Java*. Sendo assim, o *RouteFlow* ganhara suporte a mais uma tecnologia mantendo-se sempre na vanguarda das tecnologias de redes definidas por software.

**Palavras-chave**: Redes Definidas por Software

# ABSTRACT

In the last years the *OpenFlow* protocol has increased the visibility of the software defined network technologies, causing a growing number of researchers and developers to adopt it as the main tool for simulations or applications in real environments. The *RouteFlow* community project, led by the CPqD Foundation, proposes a platform defined by IP routing software based on the *OpenFlow* protocol, which allows effective separation of the control plane of routing equipment plan network. The fact that the system has been created with fully open source has caused the number of users to increase considerably encouraging the development team to update it by constantly adding more and more tools and technologies. *RouteFlow* manipulate the *OpenFlow* protocol using the most famous controllers of the literature, *NOX*, created entirely in *C++* and *POX*, created entirely in Python. To increase the capacity of the *RouteFlow*, this work describes the addiction of support for a new controller, *Floodlight*, created entirely in *Java*. Thus the *RouteFlow* win support more technology always staying at the forefront of the software defined network technologies.

**Keywords**: Software Defined Network

# LISTA DE FIGURAS


2.1 Cabeçalho *OpenFlow* para a especificação dos fluxos . . . . . 14

2.2 Exemplo de uma entrada na tabela de fluxos *OpenFlow*. . . . . 15

2.3 Exemplos de uso de um *switch OpenFlow*. . . . . 16

2.4 Rede com o protocolo *OpenFlow* habilitado. . . . . 16

3.1 Visão geral do RouteFlow. . . . . 18

3.2 Tratamento de associações do servidor RouteFlow. . . . . 23

3.3 Componentes principais do RouteFlow. . . . . 24

4.1 Esquema Geral do Proxy RouteFlow (RFProxy). . . . . 28

4.2 Fluxo das mensagens. . . . . 35

5.1 Latência com um *switch* em ms. . . . . 38

5.2 Latência com quatro *switches* em ms. . . . . 39

5.3 Desempenho com um *switch* em fluxos por segundo. . . . . 40

5.4 Desempenho com quatro *switches* em fluxos por segundo. . . . . 41

6.1 Ambiente com inúmeros proxies e inúmeras redes. . . . . 42

# LISTA DE TABELAS

3.1 Tipos possíveis de associação . . . . . 22

4.1 Representação da tabela de associação. . . . . 30

# 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*
**API** – *Application Programming Interface*
**SNMP** – *Simple Network Management Protocol*

# SUMÁRIO

**CAPÍTULO 1 – INTRODUÇÃO** **10**

1.1 Objetivo do Trabalho . . . . . 10

1.2 Contribuições . . . . . 10

**CAPÍTULO 2 – REDES DEFINIDAS POR SOFTWARE** **11**

2.1 Definição Geral . . . . . 11

2.2 Introdução ao Protocolo *OpenFlow* . . . . . 13

2.3 Descrição Geral do Protocolo *OpenFlow* . . . . . 13

**CAPÍTULO 3 – ARQUITETURA BÁSICA DO ROUTEFLOW** **17**

3.1 Introdução ao Projeto Comunitário RouteFlow . . . . . 17

3.2 Banco de Dados Centralizado com Suporte a Mecanismo de Troca de Mensagens Entre Processos . . . . . 20

3.3 Esquema de Configuração Flexível . . . . . 21

3.4 Descrições dos Principais Módulos do RouteFlow . . . . . 24

3.5 Protocolo RouteFlow . . . . . 25

**CAPÍTULO 4 – PROXY ROUTEFLOW EM JAVA** **26**

4.1 Introdução aos Proxies RouteFlow (RFProxy) . . . . . 26

4.2 Descrição Geral da Estrutura do Proxy RouteFlow em Java . . . . . 29

4.3 Descrição das Mensagens Traduzidas pelo Proxy RouteFlow . . . . . 34

**CAPÍTULO 5 – RESULTADOS** **37**

5.1 Resultados Preliminares . . . . . 37

5.2 Testes em Ambientes Virtuais . . . . . 37

5.3 Testes em Ambiente Reais . . . . . 39

**CAPÍTULO 6 – CONCLUSÃO** **42**

6.1 Trabalhos Futuros . . . . . 42

**CAPÍTULO 7 – AGRADECIMENTOS** **44**

**REFERÊNCIAS** **45**

# Capítulo 1

# INTRODUÇÃO

## 1.1 Objetivo do Trabalho

Este trabalho tem como objetivo principal agregar uma nova funcionalidade ao Projeto *RouteFlow*. A funcionalidade é 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 espera-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 utilizá-lo 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*. O uso constante do controlador *Floodlight* pelo *RouteFlow* servirá como uma plataforma de testes para o próprio controlador, contribuindo para a localização de possíveis erros ou até mesmo falta de estabilidade. 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. Apesar 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 MPLS *(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 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 indústria.

## 2.3 Descrição Geral do 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 específica 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 controlador) 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*.

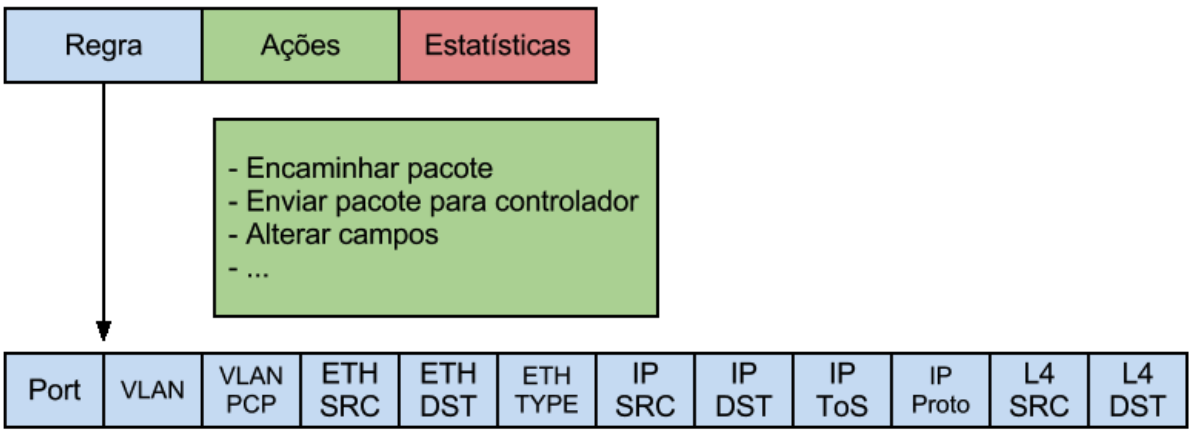


**Figura 2.2: Exemplo de uma entrada na tabela de fluxos *OpenFlow*.**

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 apresentados 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 pesquisadores 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 a redes, como um servidor comum.

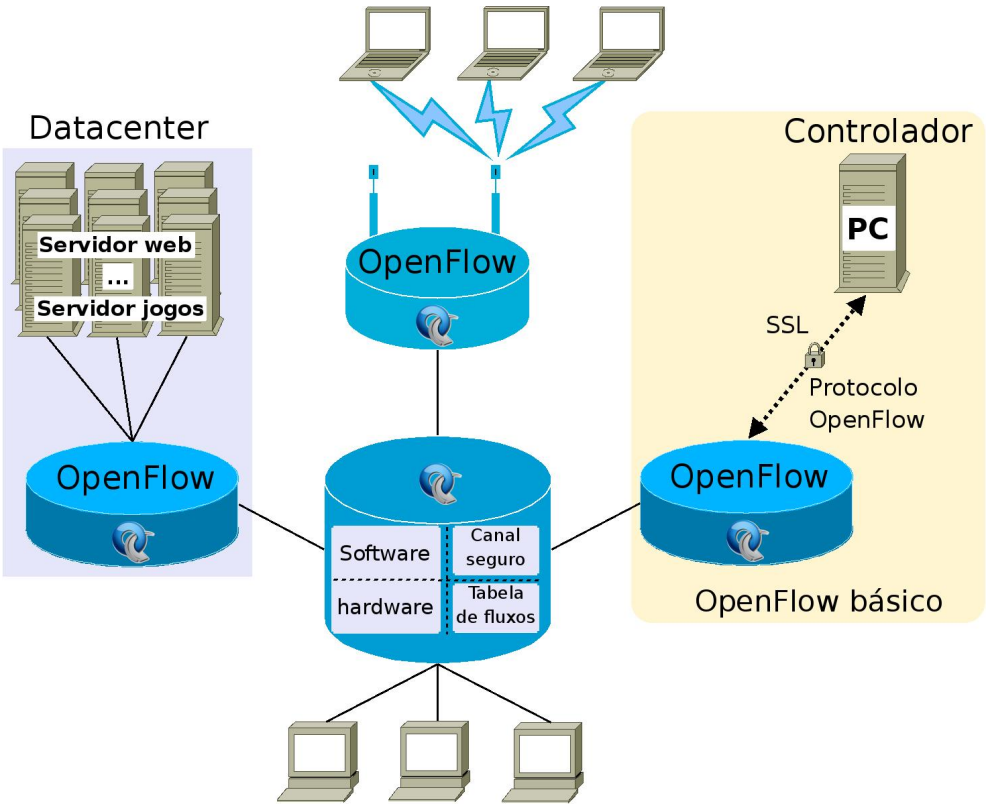


**Figura 2.4: Rede com o protocolo *OpenFlow* habilitado.**

# 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.

RouteFlow provê um serviço virtualizado de roteamento IP em equipamentos com suporte ao protocolo *OpenFlow* seguindo o paradigma das redes definidas por software. Basicamente, 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 são 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*.

RouteFlow é composto basicamente por três módulos principais: o cliente RouteFlow (RFClient), o servidor RouteFlow (RFServer) e o proxy RouteFlow (RFProxy). Todos os módulos são explicados com mais detalhes nas seções seguintes. 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 tabelas ARP. Durante a execução do ambiente virtual, as tabelas de roteamento e tabelas ARP são coletadas pelos módulos clientes (RFClient) executando nas máquinas virtuais e traduzidas em tuplas OpenFlow que são enviadas para o servidor (RFServer), que faz a adaptação das informações para o ambiente físico e finalmente instrui o módulo proxy (RFProxy), uma aplicação controladora, para configurar os *switches OpenFlow* no ambiente físico.

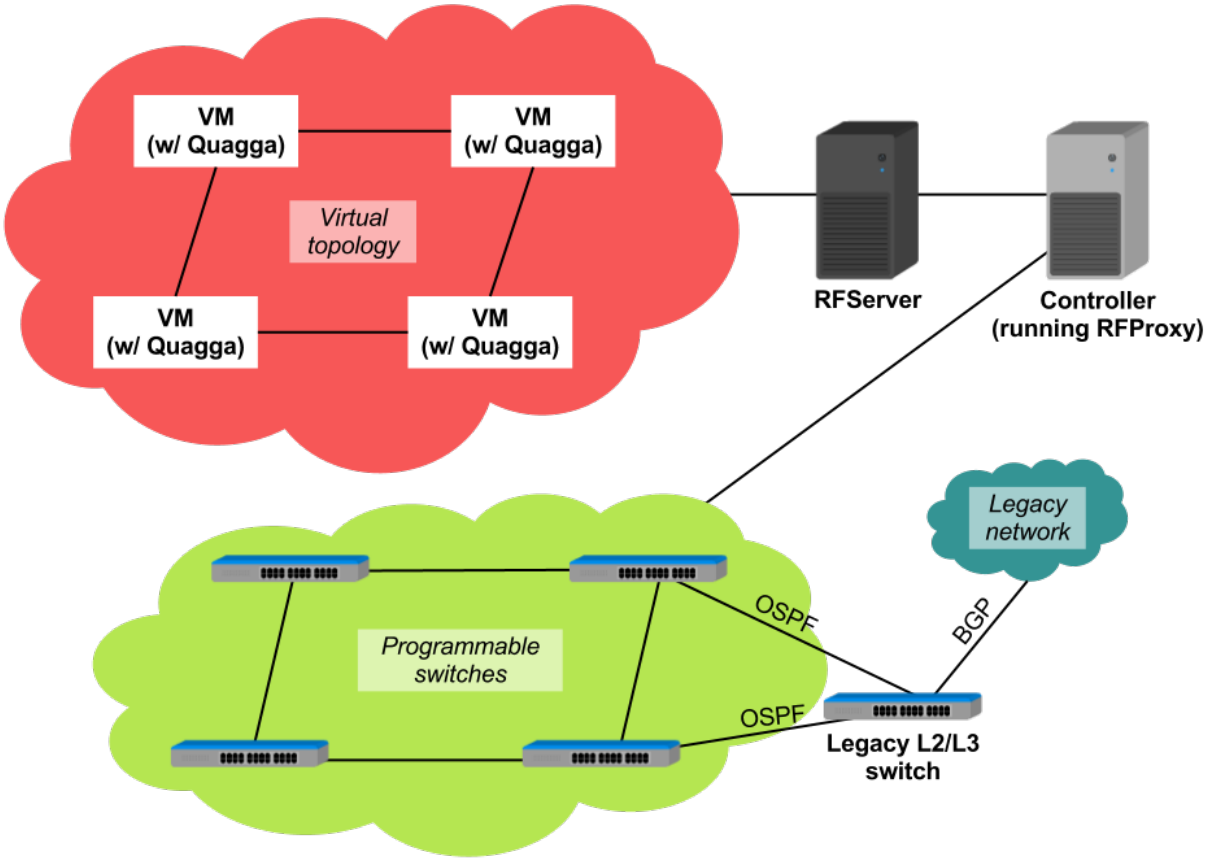


**Figura 3.1: Visão geral do RouteFlow.**

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

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 proxy RouteFlow.

O proxy RouteFlow 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 benefício 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 explicada nos itens abaixo.

- **Suporte à replicação do estado da rede e grande disponibilidade dos recursos.** Para melhorar a disponibilidade do sistema, 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 são armazenados 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. Isso aumentará a confiabilidade do ambiente.

- **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 mencionada a possibilidade de se manter um histórico das ações e decisões tomadas pelo sistema. RouteFlow, através do banco de dados centralizado, mantém 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. É possível reproduzir o sistema em ambientes em que o mesmo apresentou certa irregularidade ou falha. As estatísticas 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 a 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 Mecanismo de Troca de Mensagens Entre Processos

Nesse capítulo é explicada com detalhes a arquitetura de banco de dados centralizado e seu suporte aos mecanismos de comunicação entre processos usados pelo projeto RouteFlow. Também são abordadas as principais vantagens e desvantagens da arquitetura.

Várias abordagens foram proposta pelo projeto 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 requeriam lógica adicional para o armazenamento pendente e para mensagens já consumidas. Durante o estudo de banco de dados *No SQL* para armazenamento persistente, surgiram 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 para 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 e 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 mecanismos 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 servidor RouteFlow 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 proxy RouteFlow também podem ser armazenadas em um banco de dados centralizado, baseado em serviços adicionais é possível implementar ferramentas para análise dos dados ou 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 servidor RouteFlow em várias instâncias. A possibilidade de distribuição do servidor RouteFlow permite ao sistema um melhor desempenho, sendo possível distribuí-lo em vários pontos de uma rede e assim reduzir a latência de comunicação. Já é levado em consideração o uso de um banco de dados distribuído para as próximas atualizações do RouteFlow.

A implementação atual do RouteFlow leva em consideração as melhores técnicas e práticas usadas em aplicações executando em nuvens; isso inclui recursos de escalabilidade, banco de dados tolerante a falhas que serve como mecanismo de troca de mensagens entre processos (IPC), controle centralizado do estado da rede, armazenamento das informações usadas para desenvolvimento de aplicações de roteamento (histograma de tráfico, monitoramento de fluxos e ações administrativas). Assim podemos dizer que o banco de dados nos provê uma base com informações da rede (*Network Information Base, NIB*) [Koponen et al. 2010] e uma base de conhecimento (*Knowledge Information Base, KIB*) [Saucez et al. 2011].

## 3.3 Esquema de Configuração Flexível

Nas primeiras versões do RouteFlow, a associação entre máquinas virtuais (executando o cliente RouteFlow) e os *switches OpenFlow* era feita de forma automática e gerenciada pelo servidor RouteFlow baseada no critério de ordem de registro: o cliente de número um era associado ao primeiro *switch* à ingressar na rede, o de número dois ao segundo e assim por diante. Essa característica não requeria que o administrador controla-se o registro dos *switches* na rede mas era necessário tomar conta da ordem de ingresso dos *switches*.

Essa abordagem funcionou bem em ambientes controlados e experimentais mas apresentou problemas em ambientes onde os switches não eram controlados diretamente pelo ad-

ministrador. Para resolver este problema e ser base para o suporte ao uso de múltiplos controladores, a implementação atual do RouteFlow faz uso de um mapeamento 1:1 definido de forma manual pelo administrador da rede. É necessário o conhecimento prévio da rede pelo administrador para que o mapeamento seja feito de forma correta. Esse mapeamento é carregado e armazenado no banco de dados centralizado. A Tabela 3.1 demonstra os possíveis estados que o mapeamento pode assumir. A Figura 3.2 ilustra como o servidor RouteFlow trata os eventos da rede.

| Formato | Tipo |
|---|---|
| $vm\_id, vm\_port, -, -, -, -, -$ | Porta do Cliente Inativa |
| $-, -, -, -, -, dp\_id, dp\_port, ct\_id$ | Porta do Switch Inativa |
| $vm\_id, vm\_port, dp\_id, dp\_port, -, -, ct\_id$ | Associação entre Cliente e Switch |
| $vm\_id, vm\_port, dp\_id, dp\_port, vs\_id, vs\_port, ct\_id$ | Associação entre Cliente e Switch Ativa |

**Tabela 3.1: Tipos possíveis de associação**

Sempre que um *switch* OpenFlow ingressa na rede, o proxy RouteFlow informa ao servidor RouteFlow sobre cada uma das suas portas físicas. Cada uma dessas portas é registrada pelo servidor em um dos dois tipos mostrados na Tabela 3.1: como (i) *Porta de Switch Inativa* ou (ii) *Associação entre Cliente e Switch*. Essa associação ocorre quando não há nenhuma configuração para a porta do *switch* que foi registrada ou a porta cliente configurada para ser associada com essa porta do *switch* ainda não foi registrada. A última opção acontece quando a porta do cliente a ser associada com a porta do *switch* (baseada na configuração prévia do ambiente) está pronta e registrada como inativa.

Quando um cliente RouteFlow é iniciado, o mesmo envia ao servidor RouteFlow informações sobre suas interfaces (portas). Essas portas são registradas pelo servidor RouteFlow em um dos estados mostrados na Tabela 3.1: como uma porta de cliente inativa ou como uma associação entre cliente e *switch*. A associação é análoga à associação descrita às portas dos *switches*.

Depois da associação, o servidor RouteFlow solicita ao cliente RouteFlow para disparar a mensagem que irá confirmar que o *switch* virtual está conectado e apto a se comunicar com o proxy RouteFlow. Quando isso acontece, o proxy RouteFlow é alertado sobre a conexão entre o cliente RouteFlow e seu *switch* virtual, e envia uma informação ao servidor RouteFlow. O servidor RouteFlow decide o que fazer com a informação. Tipicamente, o proxy RouteFlow é intruído a redirecionar todo o tráfego das máquinas virtuais para os *switches* físicos associados e vice-versa. Quando um *switch* abandona a rede, todas as associações envolvendo as portas do mesmo são removidas, deixando inativas as portas do cliente envolvidas na associação. Em caso de conexão de um *switch*, o servidor RouteFlow checa se o mesmo é um novo ingres-

sante ou se é um retornante, podendo restaurar as associações configuradas previamente pelo administrador.

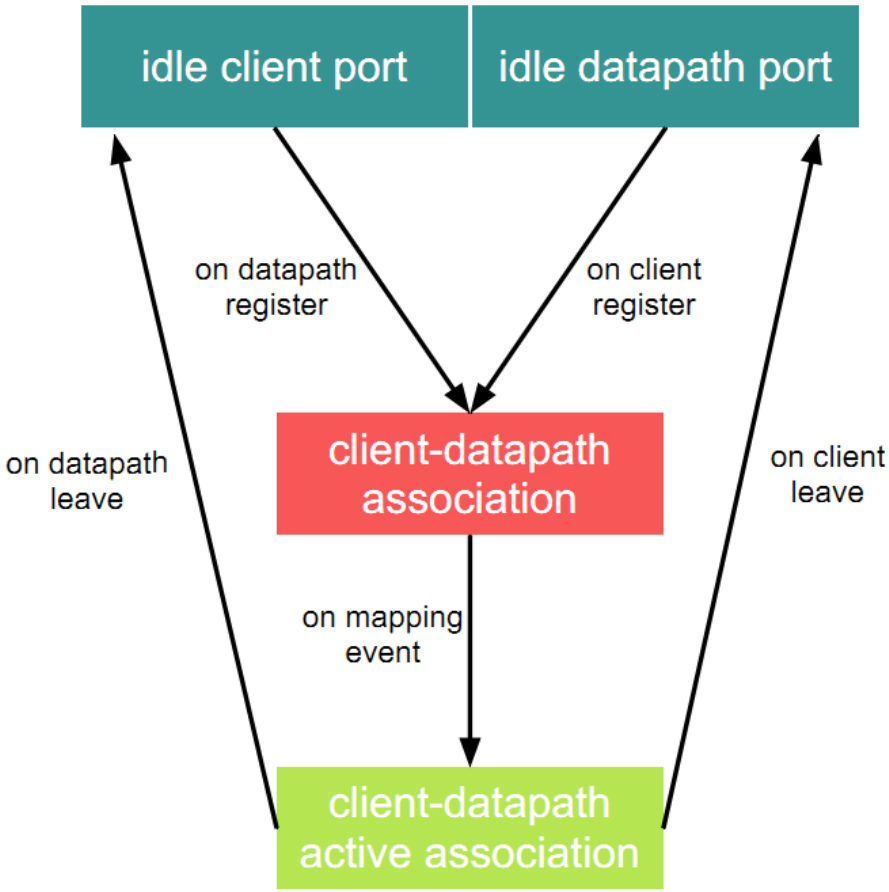


**Figura 3.2: Tratamento de associações do servidor RouteFlow.**

Uma entrada no arquivo de configuração contém um conjunto dos campos identificados na Tabela 3.1: $vm\_id, vm\_port, dp\_id, dp\_port, ct\_id$. Esses campos são suficientes para a associação, os campos remanescentes são relacionados à conexão com o *switch* virtual ($vs\_*$) e serão definidos em tempo de execução. O campo $ct\_id$ identifica com qual controlador o *switch* está conectado. Esse mecanismo permite ao RouteFlow trabalhar com múltiplos controladores, cada um controlando uma parte da mesma rede.

Considerando que o ambiente virtual pode ser distribuído, é possível executar vários domínios de roteamento com um único servidor RouteFlow, facilitando o gerenciamento de inúmeras redes roteadas em um único ponto.

## 3.4 Descrições dos Principais Módulos do RouteFlow

Como citado nas seções acima, RouteFlow é dividido em três módulos básicos: cliente (RFClient), servidor (RFServer) e proxy (RFProxy). Na Figura 3.3 há uma visão geral das aplicações:

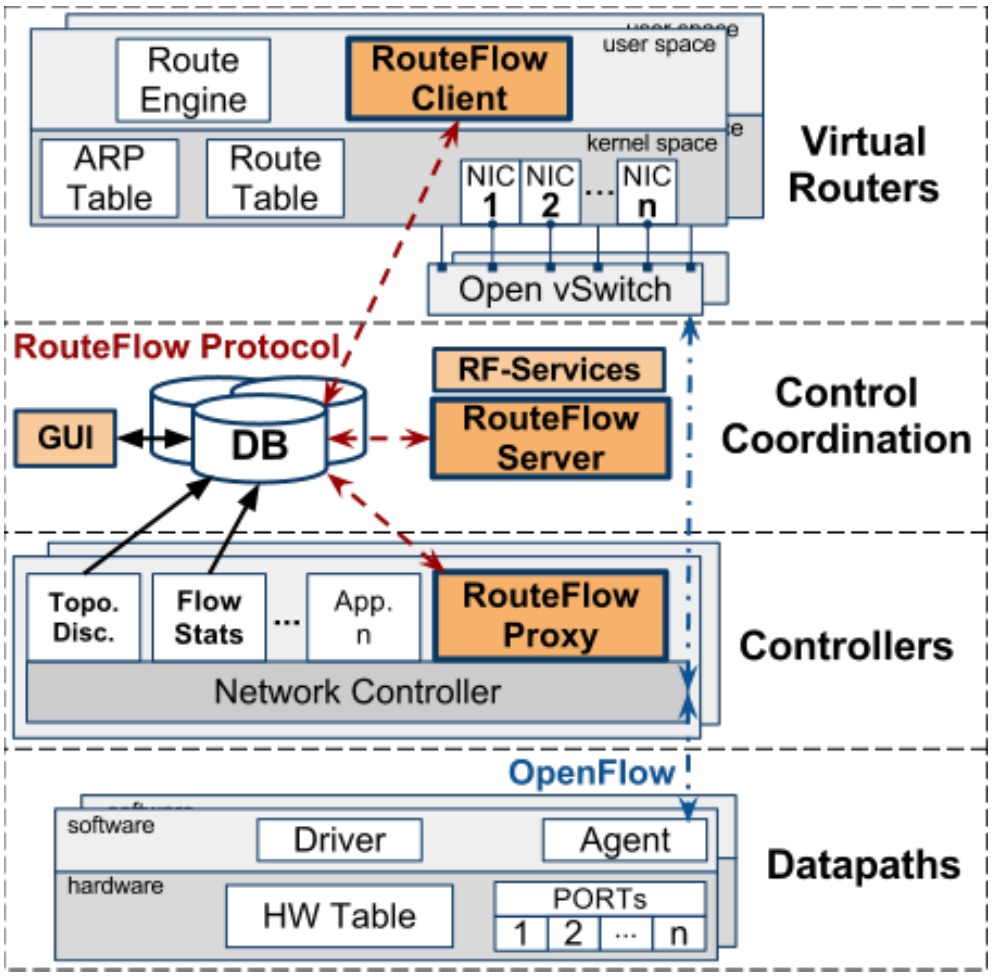


**Figura 3.3: Componentes principais do RouteFlow.**

- RFClient executa como um programa executável em um máquina virtual, detectando mudanças na tabela ARP do Linux e na tabela de roteamento. As informações sobre rotas são enviadas para o servidor (RFServer) quando são atualizadas pelas aplicações encarregadas do roteamento (QUAGGA).

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

  RFServer mantém o mapeamento entre as instâncias das máquinas virtuais executando o cliente (RFClient) e as interfaces correspondentes aos *switches* e suas respectivas portas. É conectado ao proxy (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. É considerado o módulo mais importante e mais complexo do projeto RouteFlow.

- RFProxy é uma aplicação responsável pelas interações entre os switches *OpenFlow* (identificados pelos seus datapaths) via o protocolo *OpenFlow* e os demais módulos do ambiente, como o servidor (RFServer) e o conjunto de clientes (RFClients). Ele aguarda instruções do servidor (RFServer) e o notifica à respeito de todos os eventos da rede. Atualmente é executado como um módulo vinculado aos controladores *OpenFlow*.

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

## 3.5 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.

# Capítulo 4

## PROXY ROUTEFLOW EM JAVA

### 4.1 Introdução aos Proxies RouteFlow (RFProxy)

Nos capítulos a respeito do projeto RouteFlow foi dada uma visão geral dos três módulos usados para construir o projeto. Esse capítulo se foca principalmente no módulo proxy, desenvolvido como foco principal do trabalho de conclusão de curso.

Como já dito nos capítulos anteriores, o proxy RouteFlow ou RFProxy atua como um tradutor de protocolos, convertendo mensagens no protocolo RouteFlow para o protocolo OpenFlow e vice-versa. Todos os eventos do ambiente físico são notificados ao servidor na forma de mensagens, sendo as mesmas tratadas e processadas de acordo com cada tipo de evento. Cada evento gera uma ação ao servidor RouteFlow, que responde na forma de uma mensagem ao proxy RouteFlow, instruindo-o a tomar certa atitude. Dentro dessas atitudes estão a criação das regras nos *switches OpenFlow*, fazendo toda a rede física se comportar da maneira proposta pelo ambiente virtual e seus algoritmos de roteamento.

As mensagens do protocolo RouteFlow, como já citado nos capítulos anteriores, vêm via um mecanismo de troca de mensagens entre processos (IPC) implementado através do banco de dados centralizado. Tal característica força o desenvolvedor a escolher as linguagens de programação com suporte à manipulação do banco de dados usado pelo projeto RouteFlow para criação do proxy, que no caso é o MongoDB.

As mensagens do protocolo *OpenFlow* são advindas do ambiente físico e necessitam de algum mecanismo de captura e processamento. Tal mecanismo é provido pelos softwares de controle *OpenFlow*. Dentre os diversos disponíveis, os mais famosos são NOX, POX e Floodlight. Todos os controladores *OpenFlow* são operados apenas como interfaces de programação (APIs), provendo todas as funções de captura de eventos e criação de mensagens no protocolo

*OpenFlow*.

Cada um dos controladores citados acima foi desenvolvido em uma linguagem de programação diferente, forçando o desenvolvedor a trabalhar na mesma linguagem do controlador de forma a operá-lo como forma de interface de programação (API). Cada controlador possui vantagens exclusivas, tendo sido construídos com propósitos diferentes. O controlador NOX se destaca pela performance, pelo fato de ser construído em C++ e possuir código compilado. O controlador POX se destaca pela facilidade de programação provida pela linguagem Python mas possui desempenho inferior devido ao fato do Python ser interpretado.

O controlador escolhido para uso neste trabalho foi o Floodlight. Sendo desenvolvido para ser usado principalmente por administradores de rede, possui uma série de aplicações nativas que permitem o seu controle via aplicações externas. Uma dessas aplicações nativas permite a recepção de mensagens via mensagens REST, possibilitando ao desenvolvedor manipular sua rede sem a necessidade de programar um módulo dedicado. Tal fato dificultou o desenvolvimento do trabalho, visto que grande parte da documentação disponível para consulta era voltada apenas para os administradores, sem levar em consideração os desenvolvedores de aplicações em Java. O fato de ter sido construído em Java pode impactar em seu desempenho mas tal impacto é recompensado visto o grande número de recursos disponíveis. Um dos recursos que pode ser citado é a interface gráfica usada para visualização do estado da rede que no futuro poderá ser adaptada para uso como o projeto RouteFlow. A comunidade de usuários é grande e bastante ativa, fato notado pela grande quantidade de postagens no fórum oficial do controlador e pela disposição dos principais desenvolvedores na solução de problemas encontrados ou dúvidas que surgiram ao longo do desenvolvimento.

Os dois proxies RouteFlow já existentes durante a criação desse trabalho seguem um mesmo algoritmo, o que facilita muito o entendimento geral. A maior dificuldade no desenvolvimento de um novo proxy RouteFlow são as características exclusivas de cada linguagem de programação, cabendo ao desenvolvedor as adaptações necessários para implementar o algoritmo sem grandes alterações na ideia básica. A linguagem Java já possuía uma ótima API para comunicação com o banco de dados MongoDB e para a manipulação de mensagens no protocolo JSON. Um fato que trouxe certa dificuldade no trabalho foi o desenvolvimento de uma aplicação com múltiplas threads em Java.

A Figura 4.1 ilustra de forma simples as funções básicas bem como a direção das informações tratadas pelo proxy RouteFlow. É interessante notar que o proxy só se comunica com o servidor RouteFlow através do mecanismo de troca mensagens entre processos (IPC) baseado no banco de dados centralizado.

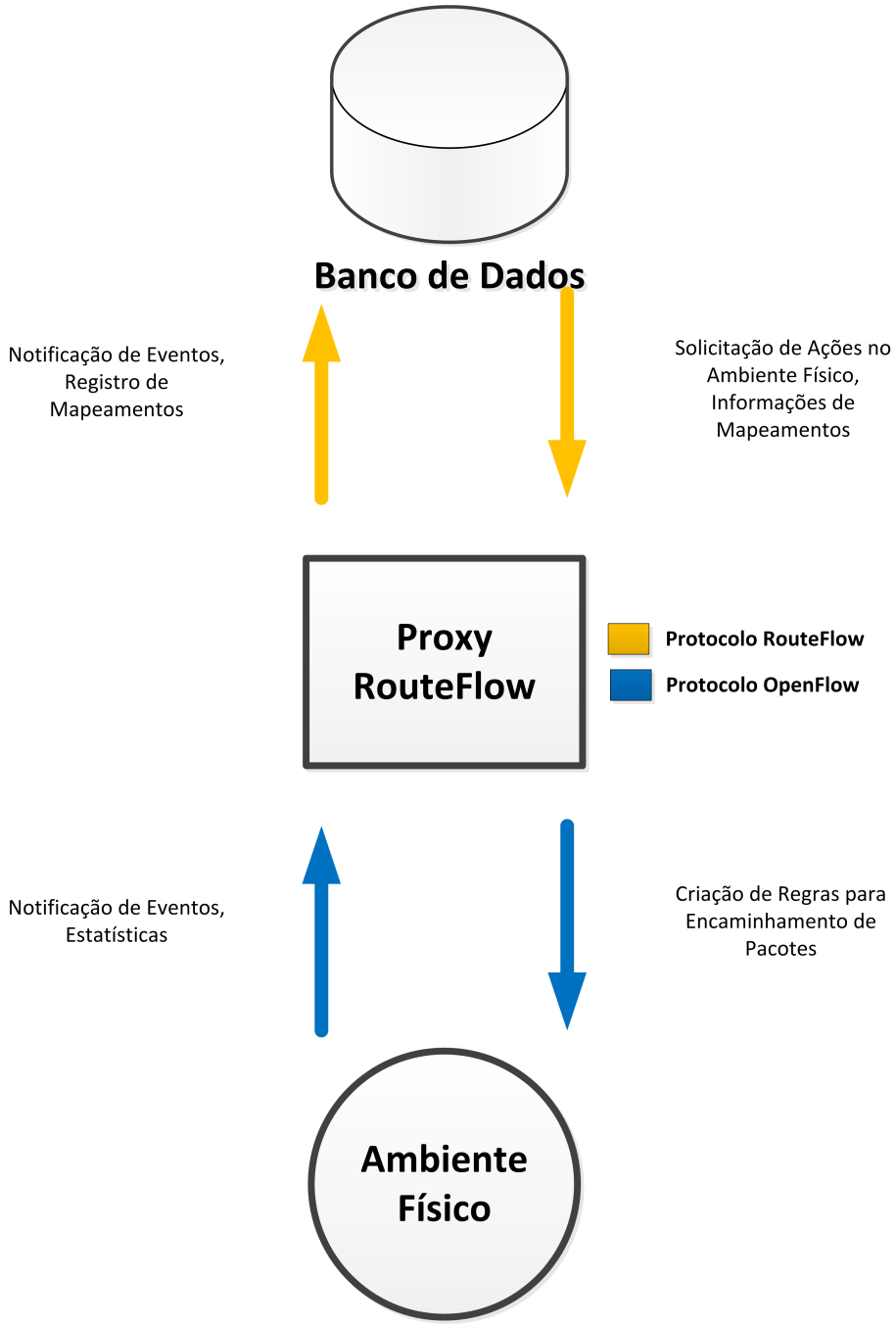


**Figura 4.1: Esquema Geral do Proxy RouteFlow (RFProxy).**

O próximo capitulo detalha as funções internas e principais estruturas do proxy RouteFlow.

## 4.2 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 facilitou a organização geral do código bem como a sua futura manutenção. Abaixo temos os principais componentes do proxy como suas respectivas 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 *No 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 mecanismo de troca de mensagens entre processos (IPC) 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 físico e virtual. 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 RouteFlow 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 gerada 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 a 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 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 portas no ambiente físico. A Tabela 4.1 nos mostra a estrutura básica da tabela de associação:

| Tipo | Função |
|---|---|
| $dp\_id$, $dp\_port$, $vs\_id$, $vs\_port$ | Interface física com interface virtual |
| $vs\_id$, $vs\_port$, $dp\_id$, $dp\_port$ | Interface virtual com interface física |

**Tabela 4.1: Representação da tabela de associação.**

Repare que a Tabela 4.1 possui redundância em suas colunas. Tal fato exige uma dupla atualização de campos sempre que ocorre uma nova associação. Isso será resolvido nas próximas versões do proxy.

No Algoritmo abaixo temos uma visão geral da ideia usada para construção do proxy, lembrando que a orientação a objetos da linguagem Java força o código executável a ser construído no formato de classe. Para deixar o código mais limpo, todas as características específicas da linguagem Java foram omitidas, deixando o código no formato de algoritmo.

```
/* Declaração dos componentes básicos.                                    */

/* Provedor de serviços OpenFlow, fornecido pelo controlador.             */
```
*floodlightProvider* ← **IFloodlightProviderService**;

```
/* Gerador de logs.                                                       */
```
*logger* ← **Logger**;

```
/* Criação do mecanismo de comunicação entre processos, será explicado com
mais detalhes.                                                            */
```
*ipc* ← **MongoIPCMessageService**;

```
/* Gerador de mensagens RouteFlow.  Cada mensagem RouteFlow tem seu tipo e
seus próprios parâmetros.  Todas as mensagens são representadas na forma de
```

```
classes; a função em questão basicamente retorna a classe representante da
função de acordo com o seu tipo.                                          */
```
*factory* ← **RFProtocolFactory**;

```
/* Processador de mensagens RouteFlow.  Será explicado com mais detalhes.  */
```
*processor* ← **RFProtocolProcessor**;

```
/* Tabela de associação.                                                   */
```
*table* ← **AssociationTable**;

```
/* Função usada para configurar regras, parcialmente fornecida pelo controlador.
*/
```
***flowConfig()***;

```
/* Função usada para deletar regras, parcialmente fornecida pelo controlador.
*/
```
***flowDelete()***;

```
/* Função usada para adicionar regras, parcialmente fornecida pelo controlador.
*/
```
***flowAdd()***;

```
/* Função usada para registrar o proxy no controlador Floodlight.  Burocracia
exigida pelo controlador.                                                 */
```
***init()***;

```
/* Função usada para o processamento dos pacotes de entrada OpenFlow.   */
```
***processPacketInMessage()***;

```
/* Função usada para iniciar a execução do proxy.  Burocracia exigida pelo
controlador.                                                              */
```
***startUp()***;

```
/* Função usada para definir quais tipos de mensagens OpenFlow serão tratadas
pelo algoritmo do proxy e especificamente por quais funções. O proxy habilita
a função processPacketInMessage a tratar as mensagens de entrada OpenFlow.
*/
```
***receive()***;

```
/* Função executada sempre que um switch OpenFlow ingressa na rede. Sempre
que isso acontece é necessário informar ao servidor RouteFlow as informações
básicas do ingressante para que o mesmo o associe a uma máquina virtual do
ambiente virtual.                                                    */
```
***addedSwitch()***;

```
/* Função executada sempre que um switch OpenFlow abandona a rede. Sempre
que isso acontece é necessário informar ao servidor RouteFlow para o mesmo
remova o registro e a associação com a máquina virtual do ambiente virtual.
*/
```
***removedSwitch()***;

Esse é o algoritmo básico do proxy RouteFlow. Todos os outros proxies implementados anteriormente seguem essa mesma arquitetura.

Abaixo temos um pseudo do mecanismo de comunicação entre processos usado para funcionamento do proxy, no algoritmo anterior a classe é chamada de **MongoIPCMessageService**:

```
/* Definição dos parâmetros de acesso ao banco de dados como nome da coleção
de dados, nome do banco de dados e endereço do banco de dados.       */

/* Função usada para criação da thread não bloqueante que executa a função
ListenWorker. A função ListenWorker é mostrada logo a seguir.        */
```
***listen()*** {
```
/* Cria a nova thread, associa a função listenWorker e inicia a sua execução.
*/
```
*};*

```
/* Função usada para checagem constante do banco de dados à procura de alguma
mensagem destinada ao processo registrado.  O processo se registra através
da criação de um canal de comunicação.  O canal de comunicação nada mais é
que uma coleção de dados no banco de dados.                              */
listenWorker() {

/* Define os parâmetros de busca.  Os principais foram definidos no início
do algoritmo.  É definido o nome da coleção que será checada de tempos em tempos
em busca de novas mensagens.                                             */

        enquanto(verdadeiro) {
                se(mensagem não lida) {
                        /* Extrai os dados importantes e altera o parâmetro lido
do banco de dados para verdadeiro.  Os dados são passados para a função processor,
mostrada no primeiro algoritmo.                                          */
                };
                senao {
                        durma
                };
        };
};

/* Função usada para envio de dados ao banco de dados.  O termo envio significa
inserir um novo conjunto de dados na coleção.  Serve basicamente para envio
de mensagens à outros processos.                                         */
send() {
};

/* Função usada para criar um novo canal de comunicação.  Usada na primeira
vez que o mecanismo de comunicação é executado.                          */
createChannel() {
};
```

Este é basicamente a estrutura básica do algoritmo usado para criação do mecanismo de comunicação entre processos. A função listenWorker, sempre que nota o recebimento de uma nova mensagem, encapsula os dados e transfere a função processor. A função processor, devido a orientação à objetos, é sobrecarregada na definição do código do proxy. Assim é usada para tomar as ações necessárias ao funcionamento do sistema. Abaixo temos um pseudo código que será mais explicado a seguir:

```
/* Função usada para tratamento das mensagens recebidas através do mecanismo
de comunicação entre processos.                                          */
processor(dados) {

        /* O parâmetro dados constitui basicamente do tipo de mensagem e seus
respectivos campos.  Para cada tipo de mensagem é necessário tomar uma ação.
*/
        caso(tipo) {
                tipo 1: /* ação 1                                          */
                tipo 2: /* ação 2                                          */
                ...
                tipo N: /* ação N                                          */
        };
};
```

Esse é basicamente o algoritmo usado em todos os proxies RouteFlow. As principais diferenças são relacionadas às linguagens de programação e ao controlador usado como interface com o protocolo *OpenFlow*.

## 4.3 Descrição das Mensagens Traduzidas pelo Proxy RouteFlow

Todas as mensagens mostradas na lista abaixo são representadas via classes na linguagem Java. As classes foram geradas automaticamente por um script em Python. Originalmente o script só fazia a geração nas linguagens C++ e Python, para que fossem utilizadas respectivamente pelos controladores NOX e POX. Foi fruto desse trabalho de conclusão de curso expandir o suporte do script para a linguagem Java. O script é construído em Python e recebe como entrada uma lista com a definição de campos e tipos de cada mensagem e retorna os arquivos com

as respectivas implementações.

A Figura 4.2 ilustra o sentido de cada mensagem, nos dizendo qual tipo de mensagem é enviado para cada módulo. Vale lembra que as mensagens são enviadas ao banco de dados centralizado para atuar no mecanismo de troca de mensagens entre processos (IPC).

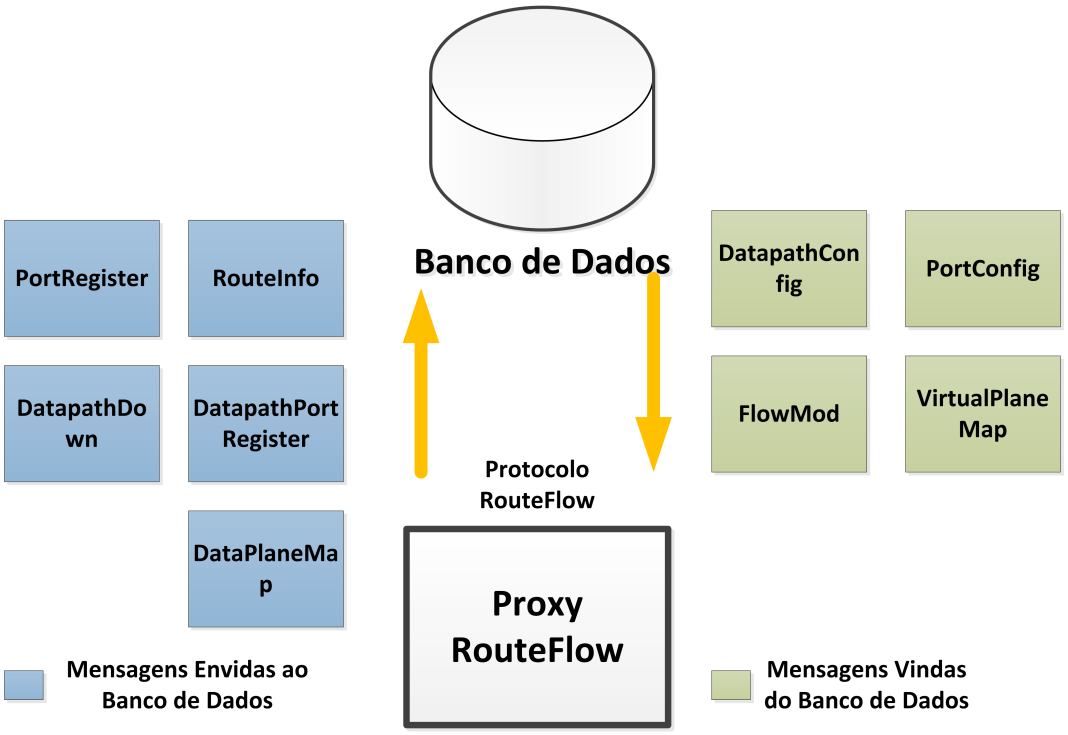


**Figura 4.2: Fluxo das mensagens.**

- *PortRegister:* mensagem utilizada sempre que um novo *switch* OpenFlow ingressa no ambiente físico. As mensagens servem para informar ao servidor RouteFlow sobre a disponibilidade de portas para que as mesmas sejam associadas às interfaces das máquinas virtuais no ambiente virtual.

- *PortConfig:* mensagem utilizada para solicitação de configuração de alguma porta do ambiente físico pelo servidor RouteFlow. Atualmente não é utilizada.

- *DatapathConfig:* mensagem utilizada para configuração dos *switches* via protocolo OpenFlow. A configuração envolve aspectos sobre como tratar certos tipos de pacotes baseado em seus protocolos.

- *RouteInfo:* mensagem não utilizada.

- *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 enviá-la corretamente ao *switch*. Para envio das regras é necessária a manipulação do protocolo OpenFlow. 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. 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 associe 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 a 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 principal resultado desse trabalho de conclusão de curso pode ser citado o suporte ao controlador Floodlight pelo Projeto RouteFlow. O Floodlight possui uma grande comunidade de pesquisa e desenvolvimento, sendo amplamente utilizado em redes de larga escala. Agora será possível aos desenvolvedores do Floodlight executarem experimentos usando o ambiente provido pelo Projeto RouteFlow.

Nas seções seguintes serão mostrados testes de desempenho e temporização de cada um dos proxies.

## 5.2 Testes em Ambientes Virtuais

Todos os testes dessa seção foram feitos em um ambiente virtual com as seguintes configurações: computador com processador Intel Core i7 2630QM e 6GB de memória *RAM DDR3*. A ferramenta de virtualização escolhida foi o VMware Player executando uma máquina virtual com quatro núcleos e 1GB de memória Ram com o Ubuntu 11.04. Os testes mostrados abaixo foram executados para todos os proxies do Projeto RouteFlow para que a virtualização tivesse o mesmo impacto sobre eles.

Para realização dos testes foi usada a ferramenta Cbench, um software específico para testes de desempenho com controladores *OpenFlow*. O programa basicamente testa os controladores, gerando eventos de entrada de pacotes para simulação de novos fluxos. O programa emula um conjunto de *switches* que se conectam no controlador, enviam pacotes de entrada (*packet-in*) e aguardam a instalação de novas regras (*flow-mods*).

Os proxies são desenvolvidos usando os controladores como interface de aplicação (API). Ao testarmos o desempenho de um controlador acoplado ao Projeto RouteFlow estaremos testando na verdade o desempenho do proxy propriamente dito. As legendas das figuras e citações abaixo fazem referência ao controlador que serviu de interface de programação ao proxy, ou seja, ao mencionarmos os controladores NOX, POX ou Floodlight estamos na verdade mencionando o proxy ao qual o controlador serviu de interface.

A Figura 5.1 mostra um teste com apenas um *switch* conectado. O ambiente montado pelo Projeto RouteFlow continha apenas uma máquina virtual executando o software de roteamento Quagga. O teste em questão leva em consideração uma distribuição cumulativa da latência. Podemos notar que o desempenho do Floodlight é parecido com o do NOX em uma grande parcela da distribuição.

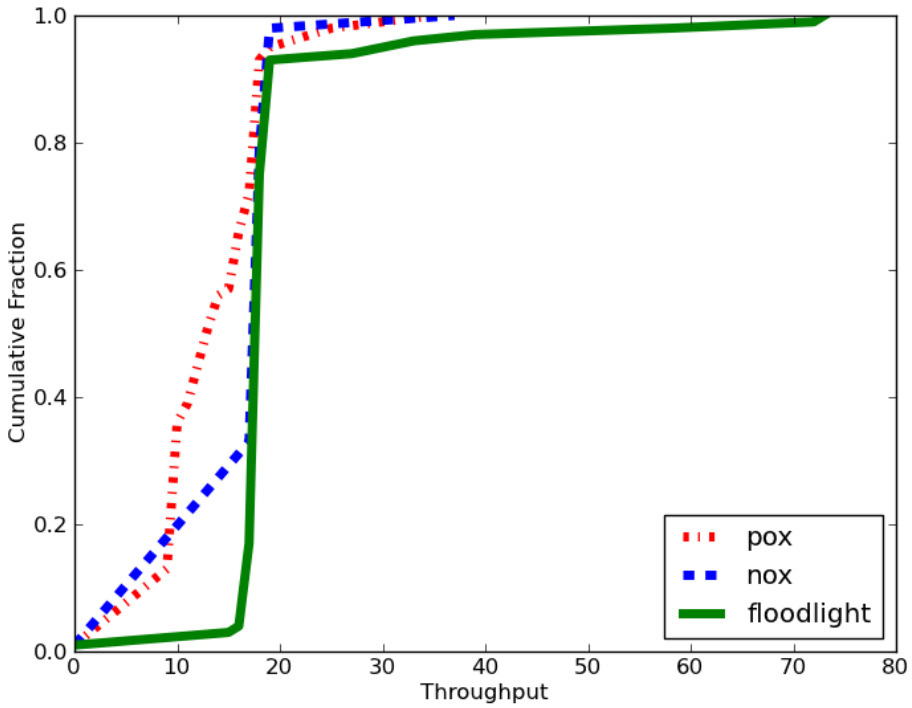


**Figura 5.1: Latência com um *switch* em ms.**

A Figura 5.2 foi obtida em um ambiente com quatro *switches OpenFlow* conectados. O ambiente RouteFlow tinha quatro máquinas virtuais executando os algoritmos de roteamento, uma para cada *switch OpenFlow*. Isso pode ter afetado seriamente o desempenho mas, para o âmbito do trabalho de conclusão de curso, estaremos testando apenas o desempenho de um proxy em relação ao outro, sem levar em consideração o real desempenho de cada um deles. Para termos dados reais de desempenho seria necessário executar todos os procedimentos em um ambiente físico. Podemos ver novamente que o desempenho do Floodlight é similar ao do

NOX em uma grande parcela da distribuição.

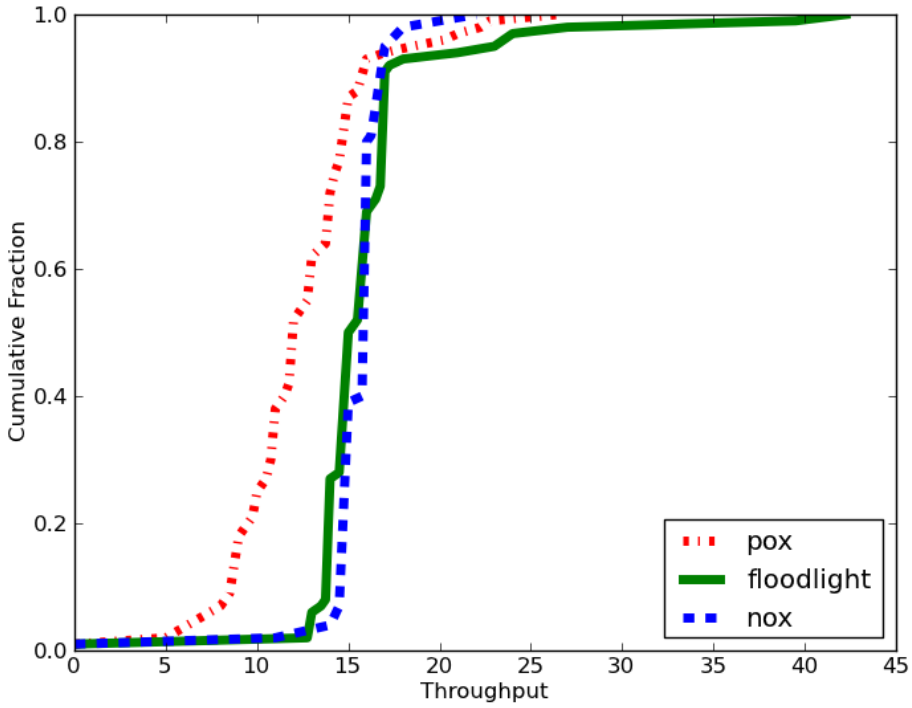


**Figura 5.2: Latência com quatro *switches* em ms.**

Os próximos testes tratam de uma distribuição cumulativa do desempenho do controlador em termos criação de regras por segundo. O ambiente é o mesmo dos testes anteriores.

A Figura 5.3 foi obtida em um ambiente com apenas um *switch OpenFlow*. Podemos observar que os três controladores tiveram desempenhos semelhantes. Tal fato pode te ocorrido devido à dupla virtualização gerada pela linguagem Java. Novas otimizações na forma de execução da linguagem Java poderão contribuir para o aumento do desempenho do Floodlight.

A Figura 5.4 foi obtida em um ambiente com quatro *switches OpenFlow*. Novamente podemos ver que o Floodlight teve desempenho parecido com o dos outros proxies, chegando a ser mais rápido que o NOX em alguns momentos.

Todos esses testes mostraram um desempenho similar entre os três proxies levados em consideração, mostrando a viabilidade de uso do proxy com Floodlight.

## 5.3 Testes em Ambiente Reais

Durante o desenvolvimento do trabalho de conclusão de curso, um pesquisador do Departamento de Tecnologia de Informação da Universidade Ghent (Bélgica) mostrou interesse

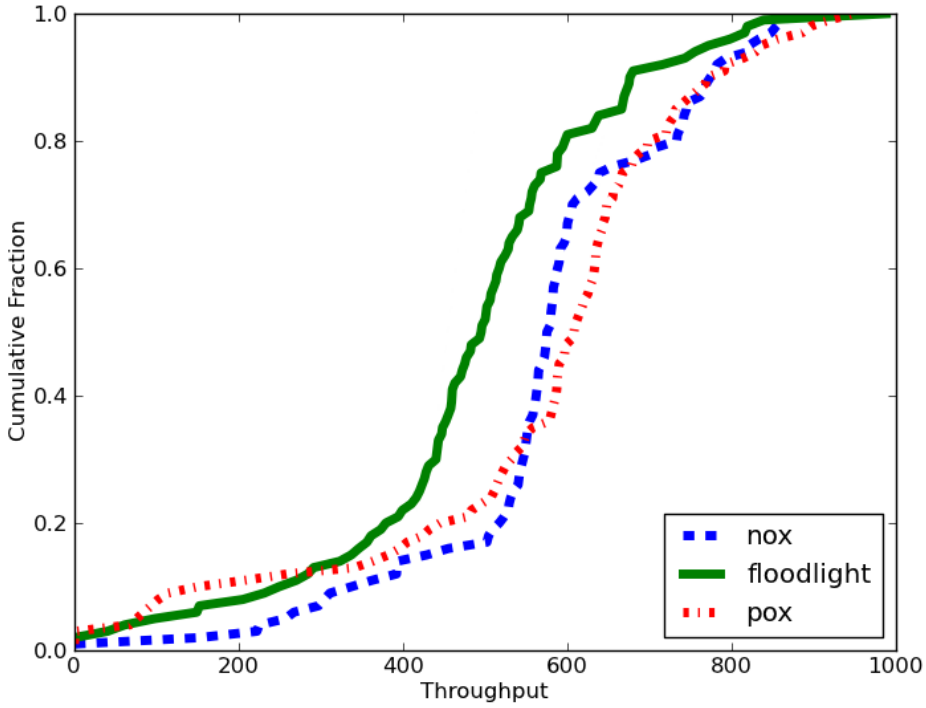


**Figura 5.3: Desempenho com um *switch* em fluxos por segundo.**

em usar o proxy RouteFlow com Floodlight para seus experimentos. Parte dos experimentos foram feitos no ambiente provido pelo Projeto OFELIA. Os experimentos foram feitos com quatro *switches OpenFlow*. Os testes iniciais mostraram que o primeiro pacote levou cerca de 30ms para chegar ao destino e os demais levaram cerca de 2ms, comprovando a instalação bem sucedida das regras pelo proxy RouteFlow. O pesquisador não relatou nenhum problema ou inconsistência na rede, mostrando a operação correta do proxy RouteFlow em Java.

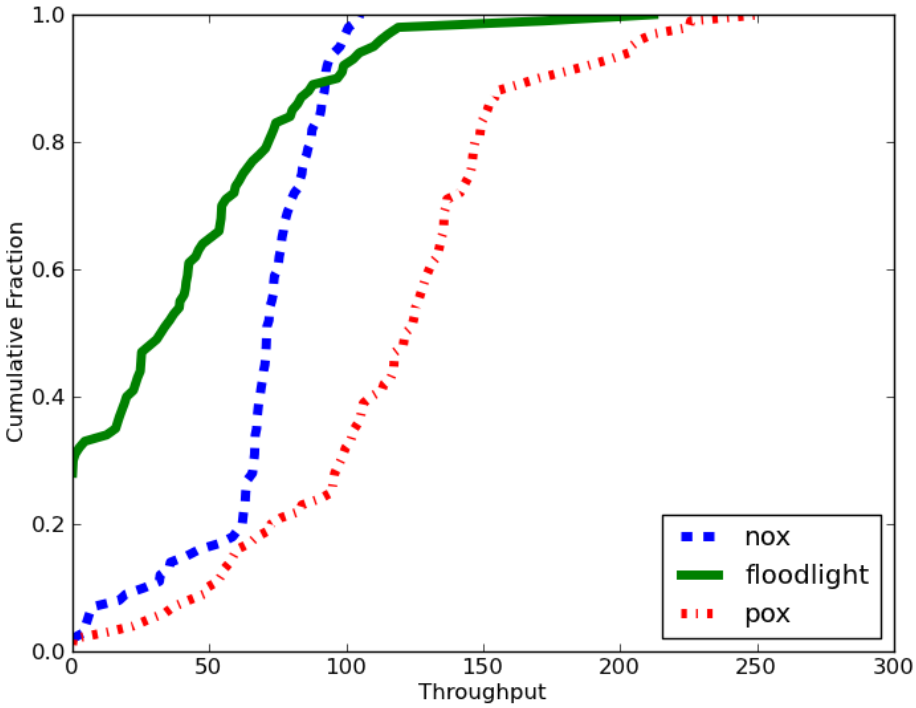


**Figura 5.4: Desempenho com quatro *switches* em fluxos por segundo.**

# Capítulo 6

# Conclusão

## 6.1 Trabalhos Futuros

Como principal trabalho futuro considera-se a manutenção e inserção de novas funcionalidades no proxy RouteFlow. Durante o desenvolvimento desse trabalho o Projeto RouteFlow ganhou suporte ao uso simultâneo de proxies, permitindo que cada trecho da rede seja controlado por um proxy diferente. Tal funcionalidade exige a inserção de mais alguns parâmetros no código original desenvolvido durante o trabalho, sendo assim o próximo trabalho futuro. A Figura 6.1 mostra de forma simples uma rede com tal funcionalidade habilitada:

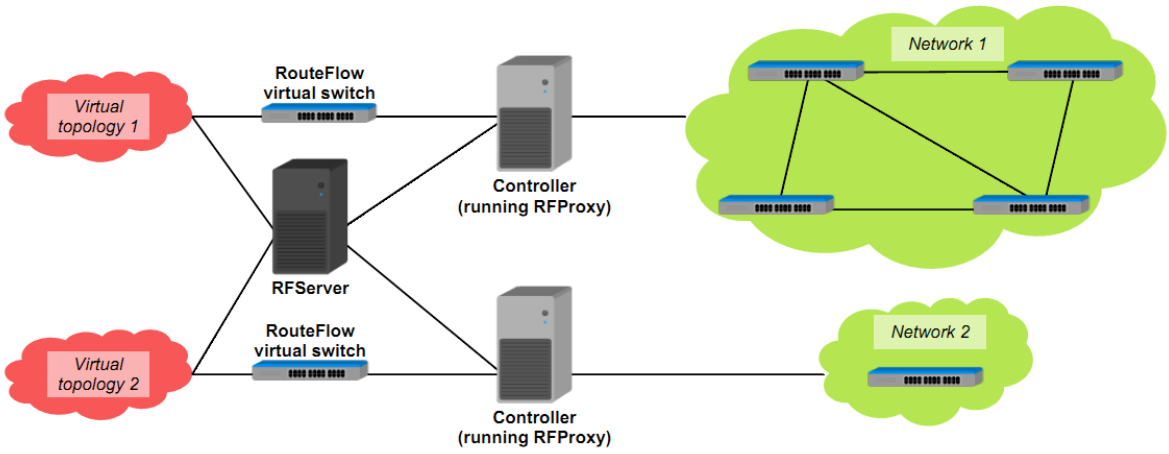


**Figura 6.1: Ambiente com inúmeros proxies e inúmeras redes.**

Outro trabalho futuro está associado à remoção do looping infinito para leitura de mensagens vindas do servidor RouteFlow. Um dos pesquisadores do Projeto RouteFlow está desenvolvendo um mecanismo de comunicação mais ágil e eficaz. Esse mecanismo também deverá ser portado ao novo proxy de forma a sempre se manter atualizado.

O site oficial do Projeto RouteFlow ainda descreve uma série de trabalhos que visam aumentar o desempenho do sistema como um todo. Muitos exigirão a manutenção e atualização do proxy. Abaixo temos uma lista das principais atualizações:

- *Multiplexação de Roteadores:* mapeamento de várias máquinas virtuais em um mesmo *switch* físico;
- *Aumento da Resiliência:* criação de um ambiente de emergência que seria ativado em caso de falha do ambiente principal;
- *Execução nas Nuvens:* execução do RouteFlow em uma nuvem pública, como a disponibilizada pela Amazon.

# 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
- Marcos Rogerio Salvador – CPqD, Campinas - SP
- Sachin Sharma – Ghent, Bélgica
- Toda a equipe do Forum do Floodlight

# REFERÊNCIAS


OPENFLOW. Innovate in your network. http://www.openflow.org/. Acessado em 10 Janeiro de 2013.

ROUTEFLOW. Virtual ip routing services over openflow networks. http://cpqd.github.com/RouteFlow/. Acessado em 10 Janeiro de 2013.

CBENCH. Scalable cluster benchmarking and testing. http://sourceforge.net/apps/trac/cbench/. Acessado em 10 Janeiro de 2013.

FLOODLIGHT. Floodlight is an open sdn controller. http://floodlight.openflowhub.org/. Acessado em 10 Janeiro de 2013.

NOX. Nox is a piece of the software defined networking ecosystem. http://www.noxrepo.org/nox/about-nox/. Acessado em 10 Janeiro de 2013.

POX. Pox is nox's younger sibling. http://www.noxrepo.org/pox/about-pox/. Acessado em 10 Janeiro de 2013.

ROTHENBERG, C. E.; NASCIMENTO, M. R.; SALVADOR, M. R.; CORRêA, C. N. A.; CUNHA DE LUCENA, S.; RASZUK, R. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In: . HotSDN '12. New York, NY, USA: ACM, c2012. p. 13–18.

TENNENHOUSE, D. L.; WETHERALL, D.; WETHERALL, D. Towards an active network architecture. In: . c2007. p. 81–94.

PETERSON, L. L.; ROSCOE, T.; ROSCOE, T. The design principles of planetlab. In: . c2006. p. 11–16.

TURNER, J. S. A proposed architecture for the geni backbone platform. In: . c2006. p. 1–10.

ELLIOTT, C.; FALK, A.; FALK, A. An update on the geni project. In: . c2009. p. 28–34.

FARRELL, R.; DAVIS, L. S.; DAVIS, L. S. Decentralized discovery of camera network topology. In: . c2008. p. 1–10.

KEMPF, J.; WHYTE, S.; ELLITHORPE, J. D.; KAZEMIAN, P.; HAITJEMA, M.; BEHESHTI, N.; STUART, S.; GREEN, H.; GREEN, H. Openflow mpls and the open source label switched router. In: . c2011. p. 8–14.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G. M.; PETERSON, L. L.; REXFORD, J.; SHENKER, S.; TURNER, J. S.; TURNER, J. S. Openflow: enabling innovation in campus networks. In: . c2008. p. 69–74.

CASADO, M.; FREEDMAN, M. J.; PETTIT, J.; LUO, J.; MCKEOWN, N.; SHENKER, S.; SHENKER, S. Ethane: taking control of the enterprise. In: . c2007. p. 1–12.

KOPONEN, T.; CASADO, M.; GUDE, N.; STRIBLING, J.; POUTIEVSKI, L.; ZHU, M.; RAMANATHAN, R.; IWATA, Y.; INOUE, H.; HAMA, T.; SHENKER, S.; SHENKER, S. Onix: A distributed control platform for large-scale production networks. In: . c2010. p. 351–364.

IANNONE, L.; SAUCEZ, D.; BONAVENTURE, O.; BONAVENTURE, O. Implementing the locator/id separation protocol: Design and experience. In: . c2011. p. 948–958.