Marcelo Frate

OrchFlow - Uma arquitetura para orquestração de redes OpenFlow com múltiplos controladores

Sorocaba, SP

23 de Fevereiro de 2017

Marcelo Frate

OrchFlow - Uma arquitetura para orquestração de redes OpenFlow com múltiplos controladores

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCCS) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Área de concentração: Engenharia de Software e Redes de Computadores.

Universidade Federal de São Carlos – UFSCar

Centro de Ciências em Gestão e Tecnologia – CCGT

Programa de Pós-Graduação em Ciência da Computação – PPGCCS

Orientador: Prof. Dr. Fábio Luciano Verdi

Sorocaba, SP

23 de Fevereiro de 2017

Frate, Marcelo

OrchFlow - Uma arquitetura para orquestração de redes OpenFlow com múltiplos controladores / Marcelo Frate. -- 2017.
99 f. : 30 cm.

Dissertação (mestrado)-Universidade Federal de São Carlos, campus Sorocaba, Sorocaba
Orientador: Prof. Dr. Fábio Luciano Verdi
Banca examinadora: Prof. Dr. Fábio Luciano Verdi, Profa. Dra. Cintia Borges Margi, Profa. Dra. Yeda Regina Venturini
Bibliografia

1. SDN - Software-Defined Networking. 2. Orquestração. 3. OrchFlow. I. Orientador. II. Universidade Federal de São Carlos. III. Título.

Ficha catalográfica elaborada pelo Programa de Geração Automática da Secretaria Geral de Informática (SIn).
DADOS FORNECIDOS PELO(A) AUTOR(A)

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências em Gestão e Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de dissertação de mestrado do candidato Marcelo Frate, realizada em 23/02/2017:

Prof. Dr. Fabio Luciano Verdi
UFSCar

Profa. Dra. Cintia Borges Margi
USP

Profa. Dra. Yeda Regina Venturini
UFSCar

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Profa. Dra. Cintia Borges Margi e, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Marcelo Frate.

Prof. Dr. Fabio Luciano Verdi
Presidente da Comissão Examinadora
UFSCar

Dedico este trabalho à minha esposa Loraine e aos meus filhos Pedro e Matheus.

Agradecimentos

Agradeço primeiramente a Deus, que sempre nos conduz vitoriosamente em Cristo e por nosso intermédio exala em todo lugar a fragrância do seu conhecimento.[1]

Agradeço também ao Prof. Dr. Fábio Luciano Verdi, pela compreensão, paciência e dedicação com a condução das pesquisas no grupo de pesquisas LERIS - Laboratory of Studies in Networks, Innovation and Software[2] e principalmente pela orientação deste trabalho.

Agradeço à minha esposa e filhos, sem os quais, eu não teria força suficiente para concluir mais esta etapa de minha vida.

Aos colegas de pesquisa do LERIS que muito contribuíram para o desenvolvimento deste trabalho, Alan Silva, André Beltrami Rocha e em especial ao amigo e colaborador Marcelo K. M. Marczuk, que através de seu trabalho de iniciação científica contribuiu ativamente para o desenvolvimento da ferramenta OrchFlow que ilustra os conceitos científicos, base das pesquisas aqui realizadas.

Por fim, um agradecimento especial ao IFSP - Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Campus Boituva[3], ao Prof. Dr. Marcelo F. Polido pela colaboração e contribuição no desenvolvimento da ferramenta OrchFlow e ao Prof. MSc. Bruno Nogueira Luz, por possibilitar através de incentivo direto esta formação acadêmica, ao qual retribuo levando os conhecimentos aqui adquiridos aos alunos dos cursos de ensino médio técnico e tecnológico nas disciplinas para a área de redes de computadores.

---

1 2 Coríntios 2:14

2 <http://leris.sor.ufscar.br/>

3 <http://btv.ifsp.edu.br/site/>

"As invenções são, sobretudo,
o resultado de um trabalho de teimoso."
(Santos Dumont)

Resumo

Desde o surgimento das Redes Definidas por Software e mais especificamente à partir de 2008 com o desenvolvimento de uma interface aberta, o protocolo OpenFlow, é possível observar que este novo paradigma de redes está revolucionando as redes baseadas no protocolo IP, possibilitando a criação de novos mecanismos de aprovisionamento de serviços, garantindo a escalabilidade e reduzindo custos. Embora este novo paradigma tenha sido criado para a centralização da lógica de controle, existe a possibilidade de descentralizá-la através da divisão das tarefas de controle entre dois ou mais controladores. Neste cenário, subdividir o domínio administrativo em subdomínios menores e fazer com que cada subdomínio seja controlado por um controlador tem sido uma alternativa para garantir escalabilidade em *Software-Defined Networking* (SDN). O protocolo OpenFlow permite a comunicação entre *switches* e controladores, entretanto ele não define como deve ser feita a comunicação de um controlador para outro controlador. Faz-se necessário, portanto, o desenvolvimento de soluções independentes do protocolo, capazes de distribuir essa lógica dentro de um mesmo domínio administrativo. Neste cenário, novas propostas vão surgindo, porém as aplicações desenvolvidas ou fazem uso de controladores iguais ou são criados novos controladores especificamente para essa finalidade. Esta pesquisa de mestrado tem como objetivo o desenvolvimento de uma arquitetura, aqui denominada de OrchFlow, capaz de receber solicitações de aplicações, orquestrando as requisições a fim de prover os serviços solicitados numa rede OpenFlow com dois ou mais controladores de implementações diferentes. A arquitetura OrchFlow, desenvolvida para esta pesquisa de mestrado, realiza essa tarefa através da orquestração de múltiplos controladores OpenFlow atuando de forma hierárquica, provendo o acesso à infraestrutura da rede através de três modos distintos: o Proativo, o Reativo e o Híbrido.

Palavras-chaves: SDN - Software-Defined Networking, OpenFlow, OrchFlow, Orquestração, Múltiplos Controladores.

Abstract

Since the emergence of the Software-Defined Networking (SDN), and, more precisely, since the development of an open interface in 2008 called OpenFlow protocol, it is being observed that this new networking paradigm is deeply remodeling the IP-protocol- based networks. It means that new mechanisms of provision services are being possible, which ensures scalability and reduces costs. Although this new paradigm has been created to centralize the control logic, there is the possibility of decentralizing it through the parceling of control tasks between two or more controllers. In this scenario, the subdivision of administrative domain in smaller subdomains in order to have each of them being controlled by one single controller has been an alternative to ensure scalability in SDN. The OpenFlow protocol allows communication among switches and controllers to another controller. However, the protocol does not define how this communication between one controller to other should be done. It is mandatory, therefore, the development of protocol independent solutions able to distribute this logic inside the same administrative domain. New proposals have been arisen, but their applications either use equal controllers or demand the development of new controllers specifically designed. This master's research aims to offer the fundamentals to the development of an architecture here so called Orch Flow, able to receive application demands and organize them in a way it provides requested services through an OpenFlow network designed with two or more different implementation controllers. The OrchFlow architecture that is being proposed accomplishes its task through handling multiple OpenFlow controllers hierarchically and providing network access through three distinct modes: Proactive, Reactive and Hybrid.

Key-words: SDN - Software-Defined Networking, OpenFlow, OrchFlow, Orquestration, Multiple controllers.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARP	Address Resolution Protocol
CPU	Central Processing Unit
DC	Datacenters
DCaaS	Datacenter as a Service
DCN	Data Center Networks
FCT	Flow Completion Time
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IP	Internet Protocol
LAN	Local Area Network
NaaS	Network as a Service
NFV	Network Functions Virtualization
NIB	Network Information Base
ONF	Open Networking Foundation
PaaS	Plataform as a Service
QoS	Quality of Service
RAM	Random Access Memory
REST	Representational State Transfer
SaaS	Software as a Service
SDN	Software-Defined Networking

SLA	Service Level Agreement
TCAM	Ternary Content-Addressable Memory
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TI	Tecnologia da Informação
UDP	User Datagram Protocol
VM	Virtual Machine
VPN	Virtual Private Network
WAN	Wide Area Network

Sumário

1 Introdução

O padrão Ethernet é uma das tecnologias de redes LAN mais populares hoje em muitos ambientes, incluindo redes empresariais e *datacenters*. Até mesmo os fornecedores de serviços de Internet usam esta tecnologia como a sua rede de *backbone* para transportar o tráfego entre vários *datacenters*. Devido ao alto desempenho para uma tecnologia de baixo custo e facilidade de configuração, quase todos os sistemas de computador hoje são equipados com uma ou várias interfaces de rede Ethernet, tornando-se a tecnologia dominante. Entretanto, este modelo limita o crescimento das redes, fazendo com que seja necessária a criação de várias sub-redes interligadas (TU et al., 2011).

Através das novas tecnologias que proporcionam a mobilidade de máquinas virtuais e a migração de aplicativos é possível se obter um balanceamento de carga de trabalho entre vários *datacenters*, gerando um uso mais eficiente dos recursos empregados. Entretanto, tais tecnologias, tornam-se dependentes da infraestrutura de rede básica para que qualquer migração de aplicativo seja bem-sucedida. É extremamente importante que as redes baseadas no protocolo IP (POSTEL et al., 1981) não ofereçam suporte apenas ao seu ambiente virtualizado conforme eles se expandem, mas que sejam também resilientes, que ofereçam alta disponibilidade e continuem evoluindo, deixando a orientação ao hardware para a orientação ao software.

Software Defined Networking (SDN) (GREENBERG et al., 2005) e *Network Functions Virtualization* (NFV) (CUI et al., 2012) começam a se firmar como o futuro das infraestruturas de rede. Tais conceitos estão mudando a forma de uso das redes pelo mundo, respondendo à crescente demanda de tráfego pelos usuários e empresas. Neste cenário, a *Cloud Computing* está revolucionando a forma como as empresas utilizam a tecnologia para potencializar seus negócios. Segundo o *National Institute of Standards and Technology* (NIST) (MELL; GRANCE, 2011):

> "*Cloud computing* é um modelo para permitir acesso ubíquo, conveniente, sob demanda de rede a um *pool* compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente fornecidos e liberados com esforço de gerenciamento mínimo ou interação do provedor de serviço."

Os *Datacenters* (DC) são responsáveis pela integração dos dados, aplicativos e recursos de TI, respondendo às tendências de tecnologia e de mercado. Para tal, são projetados para que atendam aos requisitos de desempenho em constante crescimento, vinculando-se todos os recursos do *datacenter* a uma única plataforma com um ponto

de gerenciamento central, integrando aplicativos, armazenamento, servidores e toda a infraestrutura de rede, disponibilizando acesso direto a recursos físicos e virtuais. Como explicado em (CACIATO, 2010), a consolidação do *datacenter* é uma necessidade de extrema importância para o alinhamento dos negócios e diminuição do *Total Cost of Ownership* (TCO). Através dos *Datacenter as a service* (DCaaS), softwares, plataformas, aplicativos e infraestruturas estão deixando de ser produzidos, comercializados e adquiridos apenas como produtos tornando-se serviços.

Pode-se citar entre os diversos serviços alocados em *datacenters* os *Software as a Service* (SaaS), *Platform as a Service* (PaaS), *Infrastructure as a Service* (IaaS) e *Network as a Service* (NaaS) entre outros. Destaca-se aqui o NaaS (KELLER; REXFORD, 2010), como um conceito importante, pois apresenta ao cliente a possibilidade de poupar investimento em hardware de rede e pessoal especializado, capaz de incluir o estabelecimento de rotas fim-a-fim, Rede Virtual Privada (VPN), largura de banda sob demanda, roteamento customizado, protocolos *multicast*, *firewall* de segurança, detecção e prevenção de intrusão, monitoração e filtragem de conteúdo e antivírus. Tais serviços alocados em *datacenters*, devem ser altamente resilientes, escaláveis e seguros, com o aprovisionamento dinâmico dos recursos disponibilizados em vários locais e servidores, oferecer políticas de qualidade de serviço (QoS) que priorizam o tráfego, separando-os de aplicações que disputam a largura de banda.

Para um *Datacenter*, a SDN se apresenta como uma opção para o aprovisionamento de recursos e oferta de serviços em nuvem, sob um único domínio administrativo, onde um software centraliza toda a lógica do plano de controle, definindo assim, o funcionamento lógico da rede. Um domínio administrativo compreende, portanto, toda a estrutura da rede, sob uma administração única, capaz de gerenciar todos os recursos com uma visão global e centralizada. Tal centralização, através de um controlador, possibilita uma visão global e consistente da rede permitindo a implantação de novos serviços a partir do núcleo da rede, em especial a implantação de serviços fim-a-fim.

Embora a centralização física da lógica de controle e a consolidação da infraestrutura trazidas pela computação em nuvem possam oferecer muitos benefícios, elas trazem também novos desafios com relação à administração, ao desempenho, à tolerância a falhas e à escalabilidade da arquitetura de redes para os *Cloud Datacenters*. Na medida em que os *datacenters* crescem, cresce também a quantidade de *switches* que devem se conectar ao controlador através de seu canal de controle, e como o protocolo OpenFlow (MCKEOWN et al., 2008) não especifica o número de *switches* que um controlador deve suportar, torna a escalabilidade dessa rede uma responsabilidade do servidor e de sua conexão à rede, limitando-se ao número de sessões TCP e ao desempenho da CPU.

Em resposta a tais desafios, é possível implementar um plano de controle distribuído, dividindo-se a rede em dois ou mais subdomínios, mantendo a visão global e centralizada

do domínio administrativo. No entanto, a responsabilidade do controle será dividida entre os diversos controladores OpenFlow, reduzindo a quantidade de *switches* conectados ao controlador para cada um dos subdomínios, diminuindo, portanto, a quantidade de fluxos nos canais de controle para todos os controladores. Um subdomínio faz parte do domínio administrativo e compartilha a visão de seu controlador OpenFlow com os demais subdomínios.

Isso nos remete aos mesmos desafios de escala encontrados nas redes legadas com padrão Ethernet. Essa necessidade de distribuição do controle implica em novas formas de se implementar a arquitetura SDN, quer sejam de forma distribuída ou hierárquica. Ambas as formas de administração para os diversos subdomínios administrativos ocorrem através de múltiplos controladores, implementados para atuarem de forma exclusiva ou através de controladores padrão, adaptados para que façam o compartilhamento dessa gestão. Tais soluções podem dificultar o trabalho dos administradores de rede, pois é preciso levar em consideração que a quantidade de fluxos de controle irá crescer conforme cresce a quantidade de aplicações e que tais fluxos podem sobrecarregar a capacidade de processamento dos controladores SDN.

A programação aplicada aos controladores precisa levar em consideração a demanda das aplicações, a fim de garantir que a infraestrutura e cada uma das camadas de rede estejam disponíveis de forma autônoma, oferecendo os serviços fim-a-fim necessários para que uma aplicação esteja disponível o mais rápido possível. Toda a programação deve possibilitar aos diversos controladores que mantenham os serviços ativos, respondendo aos eventos de comunicação em todo o domínio administrativo. Entretanto, para que haja melhor eficiência nos canais de controle, cada controlador é responsável apenas pelos eventos de controle que ocorram apenas no conjunto de *switches* de seu subdomínio.

A integração necessária para que dois ou mais subdomínios estabeleçam a comunicação entre dois *end-hosts* de subdomínios diferentes é de difícil administração, tornando necessário o desenvolvimento de aplicações específicas para cada situação. Aos administradores de redes, seria necessário que desenvolvessem sistemas de comunicação exclusivos à cada nova aplicação necessária na rede, programando cada controlador envolvendo não apenas os subdomínios mas também a todo o domínio administrativo. Para que todos os eventos de cada subdomínio sejam atendidos, sem prejuízo das aplicações solicitantes, torna-se necessário também a integração de eventuais aplicações que fazem uso de múltiplos subdomínios, possibilitando também a oferta de serviços fim-a-fim entre *end-hosts* de subdomínios diferentes. Para isso, é necessário o desenvolvimento de sistemas integradores que realizem a comunicação entre dois ou mais controladores, mesmo que estes sejam de implementações diferentes, proporcionando uma integração e consolidação da gerência dos diversos subdomínios para que se tenha uma visão completa da rede.

Algumas propostas sugerem que grandes redes podem ser controladas de forma

hierárquica, quer dizer, que para um domínio administrativo um controlador centralizado irá controlar os diversos controladores de subdomínios. Outras propostas admitem a distribuição em pares utilizando um protocolo de troca de tráfego entre controladores para a manutenção da completa visão da rede.

1.1 Objetivos

Esta pesquisa de mestrado tem como objetivo propor e implementar uma arquitetura para orquestração de múltiplos controladores e aprovisionamento de serviços em redes SDN baseadas no protocolo OpenFlow. A implementação desta arquitetura deu origem ao OrchFlow, que atua como um agente integrador entre as diversas aplicações disponíveis na rede e as diferentes implementações de controladores OpenFlow gerenciando diferentes subdomínios sob um mesmo controle administrativo, definido aqui como Domínio Administrativo.

O OrchFlow (FRATE; MARCZUK; VERDI, 2016) visa proporcionar ao administrador da rede uma visão global de seu domínio administrativo e um controle das ações entre todos os subdomínios, independentemente de quantos e quais controladores são utilizados. O OrchFlow visa portanto, permitir que os serviços fim-a-fim dentro do domínio administrativo continuem sendo fornecidos, assim como, serviços exclusivos podem ser criados para cada subdomínio, possibilitando entre outras coisas o estabelecimento de rotas fim-a-fim entre dois *end-hosts*, garantindo a comunicação para um tipo específico de fluxo.

O OrchFlow propõe ainda atuar de três modos: o Proativo, o Reativo e o modo Híbrido. Para todos os modos, uma determinada aplicação inicia a solicitação de um serviço ao OrchFlow, que então fará toda a orquestração necessária junto aos controladores, que encaminharão a todos os *switches* os parâmetros padronizados pela especificação OpenFlow, permitindo um controle total dos fluxos conforme as políticas do administrador do domínio.

1.2 Organização da Dissertação

Esta Dissertação está organizada da seguinte forma:

- Inicialmente é apresentado a fundamentação teórica com os conceitos básicos e os trabalhos relacionados no Capítulo 2;
- No Capítulo 3, tem-se a arquitetura e o funcionamento do OrchFlow;
- No Capítulo 4, apresenta-se os experimentos realizados e a discussão dos resultados obtidos;
- Por fim, no Capítulo 5, tem-se a conclusão desta dissertação.

2 Fundamentação Teórica

A seguir, serão abordados os pontos principais da fundamentação teórica e os trabalhos relacionados para a realização desta pesquisa de mestrado.

2.1 SDN - Software Defined Networking

O termo *Software Defined Networking* (SDN) ou Redes definidas por software define uma arquitetura programável que proporciona escalabilidade de forma dinâmica e adaptável a diversas topologias. Esta arquitetura separa o controle de rede das funções de encaminhamento permitindo o controle da rede diretamente por software, permitindo ao sistema "conhecer" toda a infraestrutura de rede.

As SDN têm recebido grande atenção nos últimos anos como forma de se resolver os desafios das redes atuais. Este novo paradigma começa a partir de duas ideias simples, generalizar hardware de rede para que ele proporcione um conjunto padrão de processamento e separar o software que controla a rede, possibilitando a evolução das redes sem a necessidade de mudança ou paralisação do hardware e permite implementar algoritmos em abstrações apropriadas para as aplicações. A abstração do controle de encaminhamento, permite aos administradores ajustar dinamicamente o fluxo de tráfego em toda a rede para atender às necessidades constantes de agilidade. O gerenciamento e a inteligência de rede são logicamente centralizados, mantendo assim uma visão global da rede e de suas políticas.

A programação da rede através de um único software permite aos gerentes de rede configurar, gerenciar, proteger e aperfeiçoar os recursos de rede mais rapidamente, possibilitando que suas aplicações sejam reescritas pois não são softwares proprietários e quando implementado por meio de padrões abertos, SDN simplifica o projeto e operação da rede pois as instruções são fornecidas por um único software em vez de vários protocolos de fornecedores de hardware. Desde o surgimento da SDN e mais especificamente à partir de 2007 com o desenvolvimento de uma interface aberta, o protocolo OpenFlow, é possível observar que este novo paradigma de redes está revolucionando as redes baseadas no protocolo IP, possibilitando a criação de novos mecanismos de aprovisionamento de serviços, garantindo a escalabilidade e reduzindo custos.

2.2 Open Networking Foundation

A *Open Networking Foundation* (ONF)[1] é uma organização dedicada à promoção e adoção e implementação de SDN por meio de padrões abertos necessários para mover a indústria de redes para essas normas. Como parte de sua missão para fazer de SDN uma realidade comercial que atenda às necessidades dos clientes, ONF está desenvolvendo padrões abertos, como o protocolo OpenFlow. Este protocolo é a primeira interface de comunicação padrão definida entre as camadas de controle e expedição de uma arquitetura SDN. Grupos de trabalho da ONF também estão abrindo o caminho para o desenvolvimento de soluções interoperáveis, colaborando com os principais especialistas do mundo em SDN e OpenFlow.

2.3 OpenFlow

Com o avanço da SDN, a padronização de elementos críticos da arquitetura SDN foi criado o protocolo OpenFlow[2], que permite a comunicação entre os planos de estruturas de controle e de dados de dispositivos de rede suportados.

OpenFlow está sendo integrado atualmente em uma variedade de dispositivos de rede e software, proporcionando benefícios substanciais para os *Datacenters*, incluindo:

- Gerenciamento centralizado e controle de dispositivos de rede de vários fornecedores;
- Melhoria de automação e gerenciamento;
- Rápida inovação;
- Programação por operadores, empresas, fornecedores de software independentes e usuários;
- Aumento da confiabilidade da rede e de segurança;
- Controle de rede mais granular;
- Melhor experiência do usuário final.

O Openflow é um padrão em desenvolvimento para a administração de Redes LAN e WAN com foco em equipamentos comerciais como *Switches*, Roteadores, *Access Points*, etc., tanto virtuais como físicos. Diversos fabricantes como Arista Networks, Big Switch Networks, Cisco, Extreme Networks, HP, IBM, Infinera, Juniper e NEC trabalham em projetos envolvendo o protocolo, inclusive alguns desses fabricantes já possuem *Switches* no mercado com suporte ao protocolo.

[1] <https://www.opennetworking.org/>
[2] <https://www.opennetworking.org/sdn-resources/openflow>

2.4 Controladores

Um controlador é basicamente um sistema criado para gerenciar e controlar as redes, oferecendo uma plataforma mediante a reutilização de componentes e a definição de novos níveis de abstração (comandos na API), pelos quais novas aplicações de rede podem ser rapidamente desenvolvidas. Este paradigma permite a evolução em paralelo das tecnologias nos planos de dados e as inovações na lógica das aplicações. As redes SDN concentram a inteligência através de um controlador, responsável por configurar todos os dispositivos da rede, manter as informações da topologia e monitorar o estado global da rede, oferecendo um controle logicamente centralizado, possibilitando a existência de um ou mais controladores físicos.

O protocolo OpenFlow define a comunicação entre *switches* e controladores, entretanto não define como deve ser feita a comunicação de um controlador para outro controlador, condição esta, necessária para qualquer tipo de distribuição no plano de controle. Existem diversas implementações de controladores OpenFlow, desenvolvidos nas mais variadas linguagens de programação, tendo em comum apenas o fato de utilizarem o protocolo OpenFlow como mecanismo de comunicação e controle dos *switches* utilizados na rede.

Alguns controladores foram desenvolvidos especialmente com o objetivo de trabalharem de forma distribuída, entretanto a maioria dos controladores atendem especificamente de forma a centralizar o controle da rede, podendo trabalhar com recursos de backup e segurança, de forma centralizada.

2.5 Trabalhos Relacionados

Atualmente, diversas propostas para as redes SDN têm feito uso de múltiplos controladores com o objetivo claro de se obter um plano de controle descentralizado. O protocolo OpenFlow a partir de sua versão 1.2, torna possível a comunicação entre os *switches* e diversos controladores, possibilitando a criação de sistemas de *backups* e redundância. Entretanto, o OpenFlow não define a comunicação necessária, entre controladores, para qualquer tipo de distribuição no plano de controle. Assim, é preciso que sejam desenvolvidas novas soluções, independentes do protocolo, capazes de distribuir essa lógica.

2.5.1 HyperFlow

O primeiro trabalho que apresentou um plano de controle distribuído para OpenFlow foi o HyperFlow (TOOTOONCHIAN; GANJALI, 2010), implementado como uma aplicação do controlador NOX, baseada em eventos, permite implantar qualquer número

de controladores e faz a distribuição do controle instanciando-se os diversos controladores, dentro de uma mesma rede OpenFlow. Cada instância é responsável por um conjunto de *switches* OpenFlow e cada *switch* é controlado por apenas um dos controladores.

Todos os controladores executam o mesmo conjunto de aplicações de controle. Entre elas está a aplicação do HyperFlow, responsável por capturar os eventos gerados em cada área da rede e divulgá-los aos demais controladores através do paradigma de publicação/assinatura. Na figura 1 cada área está representada por um site, controlada por uma ou mais instâncias do controlador.

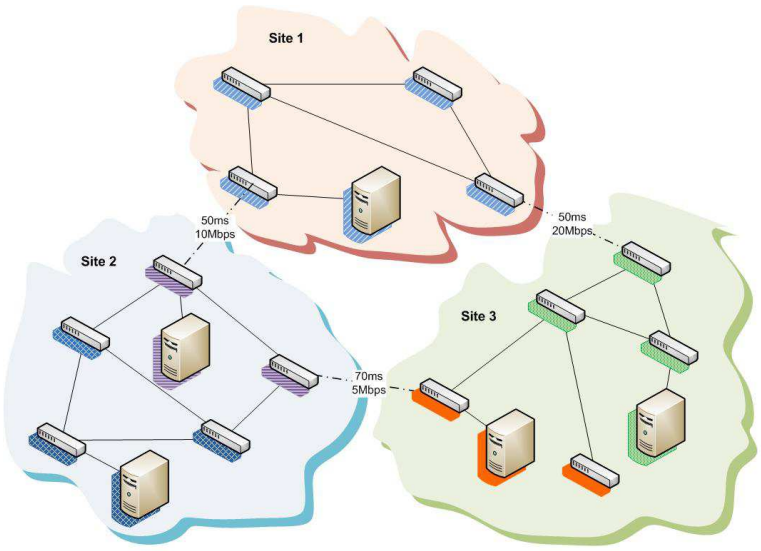


Figura 1 – Arquitetura HyperFlow.

Fonte: (TOOTOONCHIAN; GANJALI, 2010)

Este modelo permite a criação de áreas administrativas, geridas de forma independente, interligadas através de um sistema de propagação de eventos. Para realizar tal propagação, a aplicação HyperFlow usa o WheelFS (STRIBLING et al., 2009). Um sistema de arquivos distribuído projetado para o armazenamento de dados das aplicações distribuídas. Com base nesse sistema de arquivos, o HyperFlow define um diretório, como canal de divulgação de eventos e as mensagens de divulgação como arquivos. Assim, toda a tarefa de divulgação dos eventos é realizada pelo WheelFS, enquanto o HyperFlow trata do recebimento e entrega dos eventos às aplicações.

Todas as solicitações são atendidas pelos controladores locais, reduzindo o tráfego de controle interáreas, mas, mantendo atualizado cada um dos controladores por seus vizinhos, compartilhando a visão global da rede. A aplicação do controlador HyperFlow, consulta periodicamente o diretório de eventos para detectar possíveis mudanças. Na inicialização NOX, a aplicação HyperFlow inicia os serviços de cliente e de armazenamento WheelFS, assina os dados da rede e canais de controle e começa a anunciar-se periodicamente no canal de controle. O anúncio mensagem contém informações sobre o controlador, incluindo

os identificadores dos *switches* controlados por ele. Um controlador só pode programar os *switches* conectados diretamente a ele, assim, caso algum controlador falhe, outro controlador deve assumir a sua função.

Um problema que destaca-se aqui é a necessidade de se implementar uma instância do aplicativo HyperFlow em cada controlador NOX. Essa implementação requer alterações de código no núcleo do controlador, para interceptar comandos e serializar eventos.

2.5.2 DISCO

Outro trabalho que propõe a integração entre controladores é o DISCO (PHEMIUS; BOUET; LEGUAY, 2014). Tal proposta é apresentada na Figura 2. Esta proposta procura compartilhar o estado da rede entre todos os controladores, mantendo a comunicação em cascata entre controladores vizinhos.

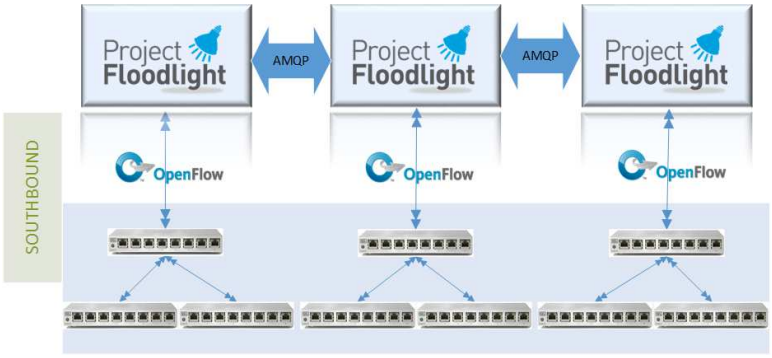


Figura 2 – Arquitetura DISCO.

Fonte: Elaborada pelo autor

A base dessa proposta está na utilização de controladores Floodlight, os quais detém as principais funcionalidades na rede. Considerando que cada controlador é responsável por um domínio e que a implementação DISCO torna possível a comunicação entre os controladores de domínios vizinhos, obtém-se assim uma visão global de toda a rede agregada através da troca de informações por parte de todos os controladores.

Esta implementação faz uso de programas agentes instalados e configurados sobre os controladores Floodlight, que permitem trocar informações entre domínios adjacentes através de um canal intercontrolador e uma interface leste-oeste *(East/West)*. Tal interface é composta por um módulo mensageiro que descobre os controladores vizinhos, fornecendo um canal de controle entre domínios utilizando um protocolo de troca de mensagens, o *Advanced Message Queuing Protocol* (AMQP) (HAT, 2012). Há diversos agentes que utilizam este canal para troca de informações com outros controladores, entre eles:

- O agente de conectividade, responsável pelo *peering*;
- O agente de monitoramento, que busca informações sobre largura de banda e latência;

- O agente de acessibilidade, que avisa caso apareça um novo *host*;
- O agente de reserva, que implementa o protocolo *Resource reSerVation Protocol* (RSVP) (BRADEN et al., 1997) para o fornecimento de recursos fim a fim.

Desta forma, cada controlador de domínio é capaz de construir uma visão da rede interdomínios e tem capacidades para realizar roteamento, reserva de caminho e gerenciamento de *Service Level Agreement* (SLA), descentralizando assim a gerência e controle da rede. No entanto, esta proposta faz uso de módulos exclusivos para o controlador Floodlight, tornando a proposta totalmente dependente deste.

2.5.3 Kandoo

Outra abordagem pode ser vista em Kandoo (YEGANEH; GANJALI, 2012), que utiliza os controladores de forma hierárquica, como pode ser visto na Figura 3, passando a existir um controlador raiz no topo da rede, interligando os diversos controladores locais, centralizando a lógica de controle e mantendo o estado de toda a rede.

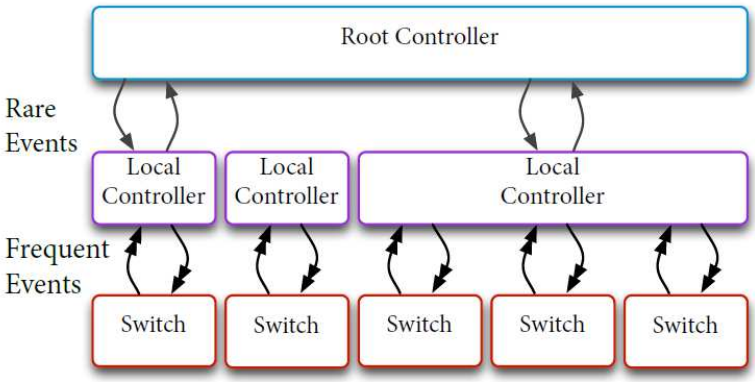


Figura 3 – Arquitetura Kandoo.

Fonte: (YEGANEH; GANJALI, 2012)

A ideia deste *framework* é viabilizar a escalabilidade sem a necessidade de mudar os controladores, formando o plano de controle distribuído com duas camadas de controle, descritas a seguir:

- *Bottom Layer*, camada em que rodam a maior parte dos eventos, apenas para as aplicações locais, onde os controladores não se interconectam e conhecem apenas parte da rede;
- *Top Layer*, um controlador logicamente centralizado, mantém o estado de toda a rede e é o responsável pela comunicação entre os diversos controladores da camada inferior.

Este modelo permite aos operadores de rede replicar controladores locais sob demanda e aliviar a carga, reduzindo o tráfego na camada de controle da rede. Aplicações locais e globais coexistem no controle da rede e as aplicações locais são as que podem operar somente com o estado local de cada *switch*, sendo implantadas, portanto, nos controladores mais próximos do plano de dados. Assim, as aplicações podem tratar mais rapidamente das requisições mais frequentes e evitam a sobrecarga no restante da rede de controle. Por sua vez, as aplicações globais são executadas por controladores no topo da hierarquia.

Cada *switch* é controlado por apenas um controlador local, que por sua vez pode controlar múltiplos *switches*. Porém, quando o controlador raiz precisa instalar um fluxo em um *switch* qualquer, ele delega as solicitações para o controlador local, responsável pelo *switch*. Para isso, cada controlador deve possuir um conjunto de protocolos próprios para a comunicação com o controlador raiz, sendo todas as comunicações entre os controladores, baseadas em eventos e de forma assíncrona.

Kandoo distribui automaticamente as aplicações de controle sem qualquer intervenção manual. A única informação extra que um controlador Kandoo necessita é uma *flag* indicando se uma aplicação de controle é local ou não. Assim, caso a aplicação seja local, nenhum tráfego será gerado até o controlador raiz. Entretanto, se a aplicação não for local, automaticamente será entregue ao controlador raiz que fará a distribuição aos controladores locais necessários. Embora esta abordagem atue de forma hierárquica, ela está restrita ao uso do controlador Kandoo e atualmente suporta apenas a versão 1.0 do protocolo OpenFlow, dificultando o desenvolvimento de novos experimentos.

2.5.4 Onix

Controladores distribuídos como em Onix (KOPONEN et al., 2010), sendo executado em um *cluster* de um ou mais servidores físicos, podem ser observados na Figura 4, utilizando um modelo de dados chamado *Network Information Base* (NIB). Este modelo mantém o estado da rede através de uma estrutura de dados que armazena um grafo de todas as entidades da topologia. É através da replicação e distribuição da NIB entre as múltiplas instâncias que o Onix consegue prover a escalabilidade.

A utilização do Apache ZooKeeper (HUNT et al., 2010), uma *Application Programming Interface* (API) *Open Source* que permite que os processos distribuídos sincronizem informações um com o outro sem falha, de modo que todos os clientes que fazem solicitações recebam dados consistentes, possibilita ao Onix manter informações de configuração, nomeação de servidores, provimento de serviços de grupos e provimento de sincronização distribuída.

Onix é responsável por habilitar a lógica de controle das aplicações, operar sobre o

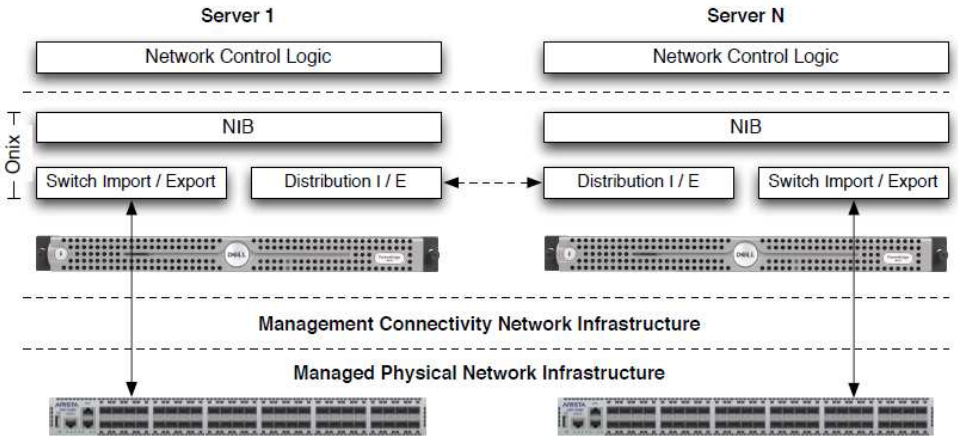


Figura 4 – Arquitetura Onix.

Fonte: (KOPONEN et al., 2010)

estado dos elementos de rede e por disseminar o estado da rede para outras instâncias dentro de um *cluster*. Essa lógica é codificada sobre a sua API que determina o comportamento da rede, suportando três estratégias que podem ser usadas para melhorar a escalabilidade dos sistemas de controle:

- Particionamento: uma instância do controlador mantém apenas um subconjunto da NIB na memória;
- Agregação: uma instância pode expor um subconjunto de elementos em sua NIB como um elemento agregado para outras instâncias;
- Consistência e Durabilidade: a lógica de controle determina os requisitos de consistência para a rede que ela gerencia.

A NIB é o ponto de integração central entre outras instâncias Onix e outros elementos de rede. Eles importam e exportam os estados para a NIB. Cada aplicação deve declarar quais dados usar e devido às possíveis inconsistências que isso possa gerar, a Onix espera que as aplicações registrem a sua lógica de inconsistência na plataforma, através de entidades herdadas ou por *plug-ins*.

A Onix tem uma contribuição relevante, mas, mesmo tendo por base a utilização de softwares de código aberto como o controlador NOX e do Apache ZooKeeper, foi desenvolvido em código fechado, o que impossibilita a integração e o desenvolvimento de novas aplicações.

2.5.5 Comparações

Outros trabalhos encontrados na literatura tem por objetivo proporcionar a escalabilidade das redes de transporte comutada por pacotes, sendo que alguns levam em

consideração e permitem o uso do protocolo OpenFlow, por se tratar de um novo paradigma para as redes atuais. Entretanto, nesta dissertação, optou-se pelo uso exclusivo do protocolo OpenFlow, não levando em consideração as possibilidades que outros protocolos possam oferecer.

Tabela 1 – Tabela Comparativa.

Arquitetura	**Controladores**	**Comunicação**	**Hierárquico**
HyperFlow	NOX	WheelFS	NÃO
Disco	Floodlight	AMQP	NÃO
Kandoo	Kandoo	Protocolo Proprietário	SIM
Onix	NOX	Zookeeper	NÃO
OrchFlow	Diversos	Interface REST	SIM

Como é possível observar na Tabela 1, algumas propostas buscam desenvolver os seus próprios controladores, como o Kandoo e o Onix, entretanto, Kandoo faz uso de protocolo próprio de comunicação para as interfaces leste-oeste, enquanto Onix utiliza um protocolo de código aberto, o Zookeeper.

O HyperFlow utiliza o controlador NOX de código aberto e usa programas agentes instalados e configurados a fim de criar um mecanismo de troca de informações entre eles, modificando o funcionamento padrão do controlador utilizado.

Disco faz uso de um controlador de software aberto, o Floodlight, entretanto, utiliza o protocolo AMQP para a distribuição dos dados e não é hierárquico.

Não havia até o presente momento, uma proposta que fizesse uso de diferentes controladores. Assim, O OrchFlow surge como uma solução, possibilitando o uso de diversas implementações de controladores, utilizando uma interface REST para a comunicação entre os controladores e o OrchFlow, possibilitando ainda o funcionamento de forma hierárquica.

3 OrchFlow

Neste capítulo, detalhamos o funcionamento do OrchFlow e dos seus três modos de atuação: o proativo, o reativo e o híbrido.

3.1 A ferramenta

As redes SDN são baseadas na concentração de sua inteligência através de um software controlador e que tal software oferece um controle logicamente centralizado. Entretanto, a partir da versão 1.2 do protocolo OpenFlow, é possível a existência de outros controladores atuando como *slave* substituindo o controlador *master* em caso de falha.

Tais configurações oferecem maior confiabilidade à rede, pois em caso de perda de conexão com o controlador principal um controlador *slave* passaria a responder a todos os eventos. Porém, há ainda, diversas razões para se usar um plano de controle distribuído, tais como: administração, distribuição geográfica, escalabilidade, redução de latência entre um *switch* e seu controlador, tolerância a falhas e balanceamento de carga. Assim, apesar de haver a necessidade de subdividir a topologia para que cada parte seja gerenciada de maneira independente, criando subdomínios de controle, há ainda a necessidade de gerenciamento e oferta de serviços globais que envolva todo o domínio administrativo.

Desta forma, o OrchFlow se apresenta como uma ferramenta que tem como objetivo funcionar como um *Middleware*, um software orquestrador para as redes SDN baseadas no protocolo OpenFlow. Ele pode receber solicitações de serviços através de uma interface Norte *(Northbound)*, processá-las e então mapeá-las através de uma interface Sul *(Southbound)*, provendo o serviço solicitado através de múltiplos controladores. Quando um determinado serviço é solicitado, o OrchFlow define como deve ser o tratamento por parte dos diversos controladores até que sejam aplicadas as regras OpenFlow a todos os elementos de rede sob o mesmo domínio administrativo.

3.1.1 Funcionamento do OrchFlow

Para melhor compreensão da arquitetura proposta e do seu funcionamento, pode-se observar na Figura 5 que o OrchFlow atua como um agente integrador entre as diversas aplicações disponíveis na rede e as diferentes implementações de controladores OpenFlow, que gerenciam os diversos subdomínios sob um mesmo domínio administrativo.

Na base desta arquitetura estão os *switches*, formando toda a infraestrutura necessária para a comunicação entre os *end-hosts* e para cada subconjunto de *switches*, definidos aqui como subdomínio, deve-se ter um controlador responsável por toda a configuração

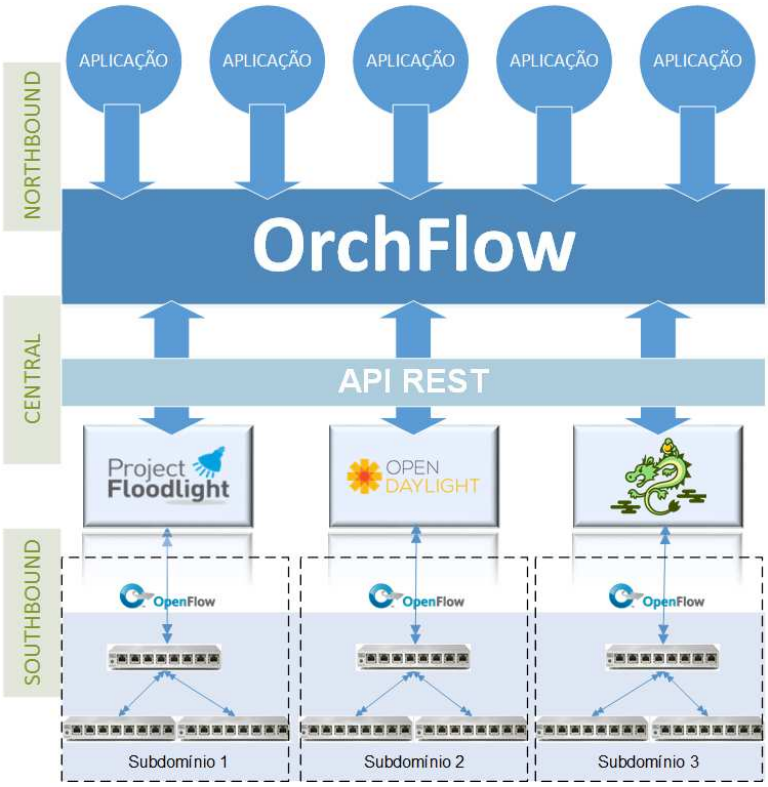


Figura 5 – Arquitetura OrchFlow.

Fonte: Elaborada pelo autor

e manutenção de todas as informações da topologia, assim como pelo monitoramento de todo o estado desta rede. Tais configurações se dão através da interface Sul e do protocolo OpenFlow, onde uma API realiza a comunicação entre os elementos de controle (Controladores) e os elementos de encaminhamento de dados (*Switches*).

Cada controlador está conectado ao OrchFlow através de uma interface Central, uma interface de integração responsável pela comunicação entre as diferentes implementações de controladores e o OrchFlow, que atuará diretamente na orquestração destes, independentemente de quais linguagens de programação os controladores foram implementados.

O OrchFlow possibilita também a comunicação entre as diferentes aplicações através de uma interface Norte, capaz de receber solicitações específicas e que podem invocar serviços pré-determinados pelo OrchFlow. Essas interfaces fazem parte de um único software, que permite a invocação de serviços através de uma interface *Representational State Transfer* (REST) (FIELDING, 2000), pela qual são aplicadas as regras estabelecidas para cada aplicação solicitante, orquestradas e entregues a cada um dos controladores, conforme o subdomínio a ser alcançado.

O OrchFlow proporciona ao administrador da rede uma visão global de seu domínio administrativo e um controle total das ações entre todos os subdomínios, independente-

mente de quantos e quais controladores são utilizados. O OrchFlow permite, portanto, que os serviços fim-a-fim dentro do domínio administrativo continuem sendo fornecidos, assim como serviços exclusivos podem ser criados para cada subdomínio, possibilitando entre outras coisas o estabelecimento de rotas fim-a-fim entre dois *end-hosts*, garantindo a comunicação para um tipo específico de fluxo. De forma mais específica, o OrchFlow é capaz de receber solicitações de serviços através de uma interface Norte a fim de processá-las e então mapeá-las através de sua interface Central, de forma a prover o acesso à infraestrutura necessária para que tal serviço seja provido em uma rede controlada por múltiplos controladores com diferentes implementações.

O OrchFlow fornece três modos de atuação: o Proativo, o Reativo e o Híbrido.

Proativo: Como mostrado na Figura 6, toda a programação necessária para o estabelecimento de serviços fim-a-fim é executada pelo OrchFlow e transferida aos controladores através do método applyRule(). Ao receber a programação, cada controlador aplica imediatamente todas as regras diretamente nas tabelas de fluxos dos *switches* envolvidos através de mensagens Flowmod(). Os parâmetros a serem enviados aos *switches* são aqueles padronizados pela especificação OpenFlow, permitindo assim, um controle total dos fluxos conforme as políticas do administrador do domínio.

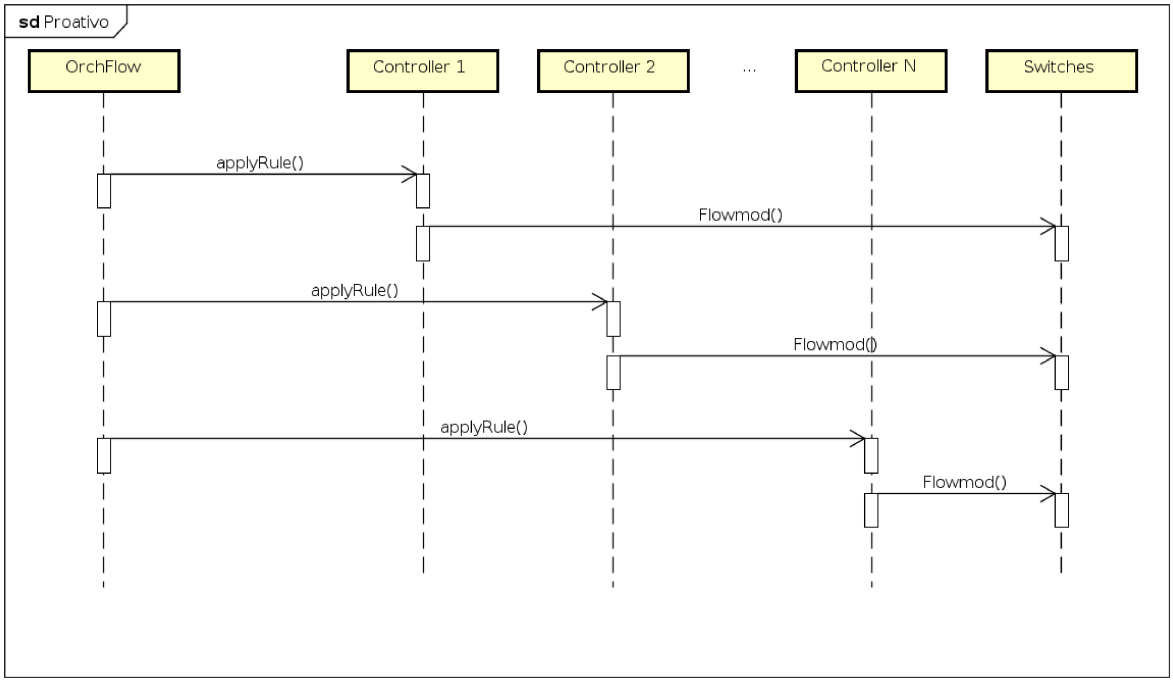


Figura 6 – Diagrama de Sequência - Modo Proativo.

Fonte: Elaborada pelo autor

Assim, quando um novo fluxo de dados chegar na rede OpenFlow, este será encaminhado diretamente sem gerar um evento *packet-in*. Este modo oferece aos fluxos o encaminhamento imediato e, mesmo que haja a perda de conexão com o controlador, os

fluxos não serão descartados pois as regras estarão previamente implantadas nos *switches*. Tais regras passam a ocupar a memória TCAM em cada um dos *switches* e permanecem nela pelo tempo estabelecido através da aplicação solicitante.

Reativo: No modo reativo, como mostrado na Figura 7, o OrchFlow encaminha as regras para cada um dos controladores através do método sendReactiveData(), porém, ao invés destas regras serem imediadamente implantadas nos *switches* (como é feito pelo modo proativo), elas serão armazenadas em uma tabela de nome *Reactive*, criada através de um módulo de mesmo nome previamente instalado em todos os controladores.

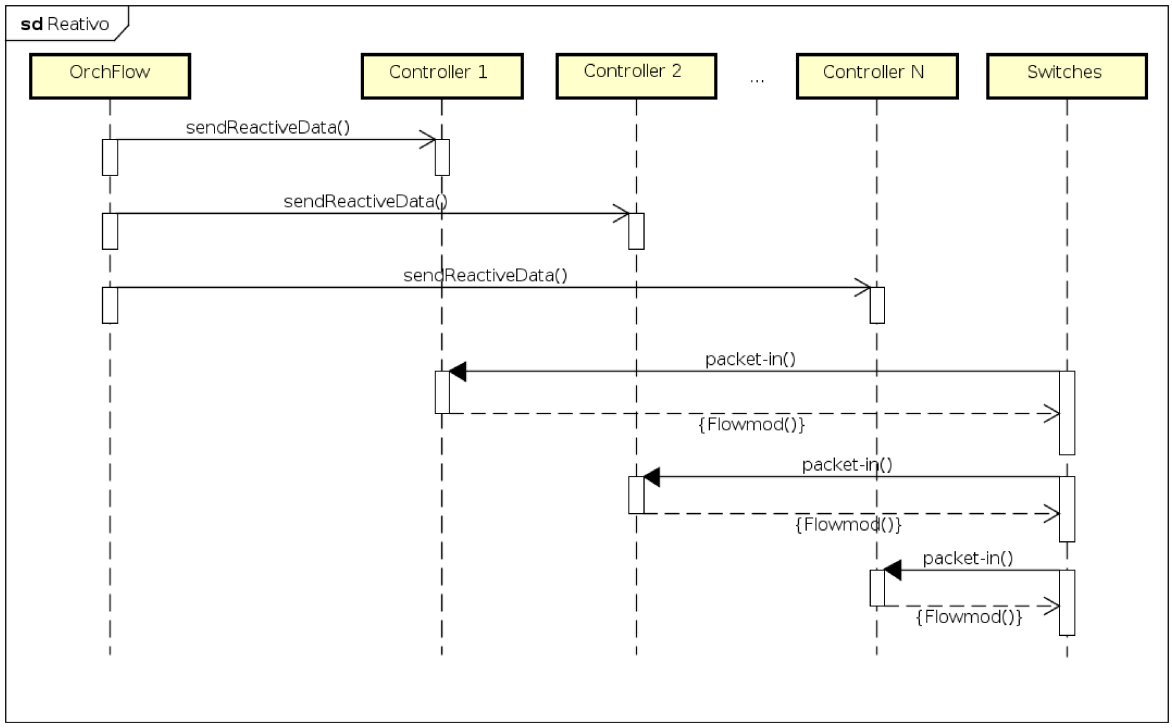


Figura 7 – Diagrama de Sequência - Modo Reativo.

Fonte: Elaborada pelo autor

Deve-se observar aqui, que ao receber um fluxo, um *switch* de borda verifica em sua tabela de fluxos a existência de uma correspondência única ao cabeçalho do fluxo recebido. Caso não encontre, o *switch* dispara um evento *packet-in* para o seu controlador, através da interface Sul (OpenFlow). Ao recebê-lo, o controlador fará uma comparação com os dados de sua tabela *Reactive* e, caso haja uma correspondência, adiciona através de mensagens Flowmod() as regras nas tabelas de fluxos de todos os *switches* de seu subdomínio.

Como mostrado na Figura 8, apenas o primeiro pacote de um fluxo é que faz com que o *switch* dispare o evento *packet-in* ao controlador e este deverá responder com uma regra pré-estabelecida pela aplicação solicitante e implantá-la em todos os *switches* envolvidos na comunicação fim-a-fim de seu subdomínio. Note também que este processo se repete nos próximos subdomínios até que o serviço fim-a-fim solicitado seja estabelecido

em sua completude.

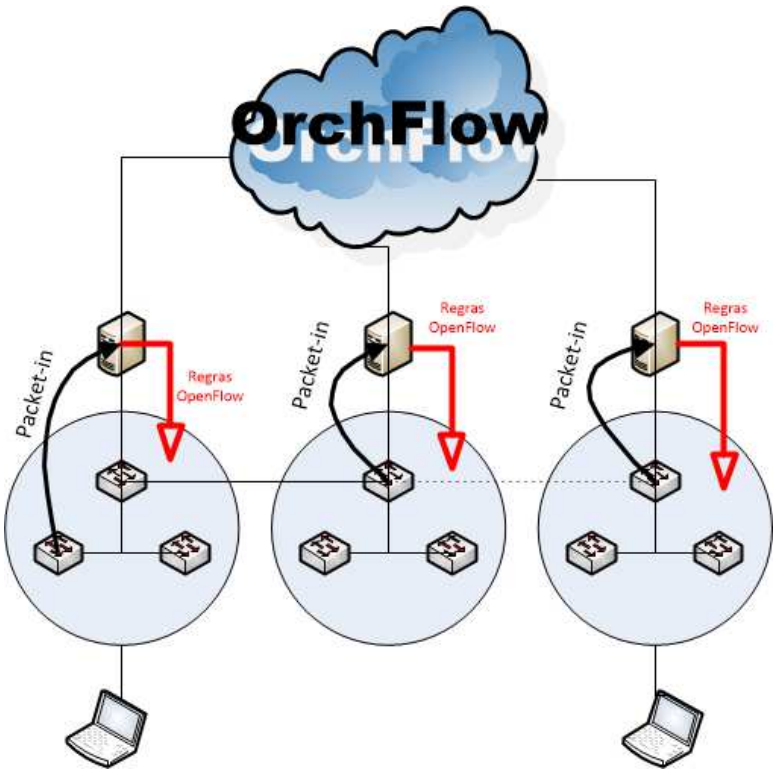


Figura 8 – Modo Reativo.

Fonte: Elaborada pelo autor

De modo geral as regras criadas pelo modo reativo recebem um tempo de vida para que as regras expirem, ocupando assim espaço na memória dos *switches* por um tempo limitado às necessidades de cada aplicação.

Híbrido: O modo híbrido funciona praticamente de forma idêntica ao reativo, ou seja, as regras são criadas e enviadas através do método sendReactiveData() para os controladores mas não são implantadas imediatamente nos *switches*. Como mostrado na Figura 9, a diferença em relação ao modo reativo ocorre quando o evento de *packet-in* for recebido pelo controlador do primeiro subdomínio (por onde o fluxo entra na rede). Neste caso, o controlador do subdomínio, ao encontrar uma correspondência para tratar o fluxo, envia uma notificação através do método hybrid() ao OrchFlow sobre o evento. O OrchFlow, por sua vez, notificará através do método applyRule() os controladores dos próximos subdomínios para que as regras correspondentes sejam imediadamente implantadas em seus subdomínios, de forma a se antecipar à chegada do fluxo de dados.

Este mecanismo, como pode ser observado na Figura 10, atua inicialmente de maneira reativa mas que, após o primeiro evento de *packet-in*, atua de maneira proativa. No melhor caso, nenhum outro evento de *packet-in* será disparado nos próximos subdomínios, diminuindo assim o tempo de transferência total dos dados. No pior caso, o modo híbrido se comportará de maneira idêntica ao modo reativo.

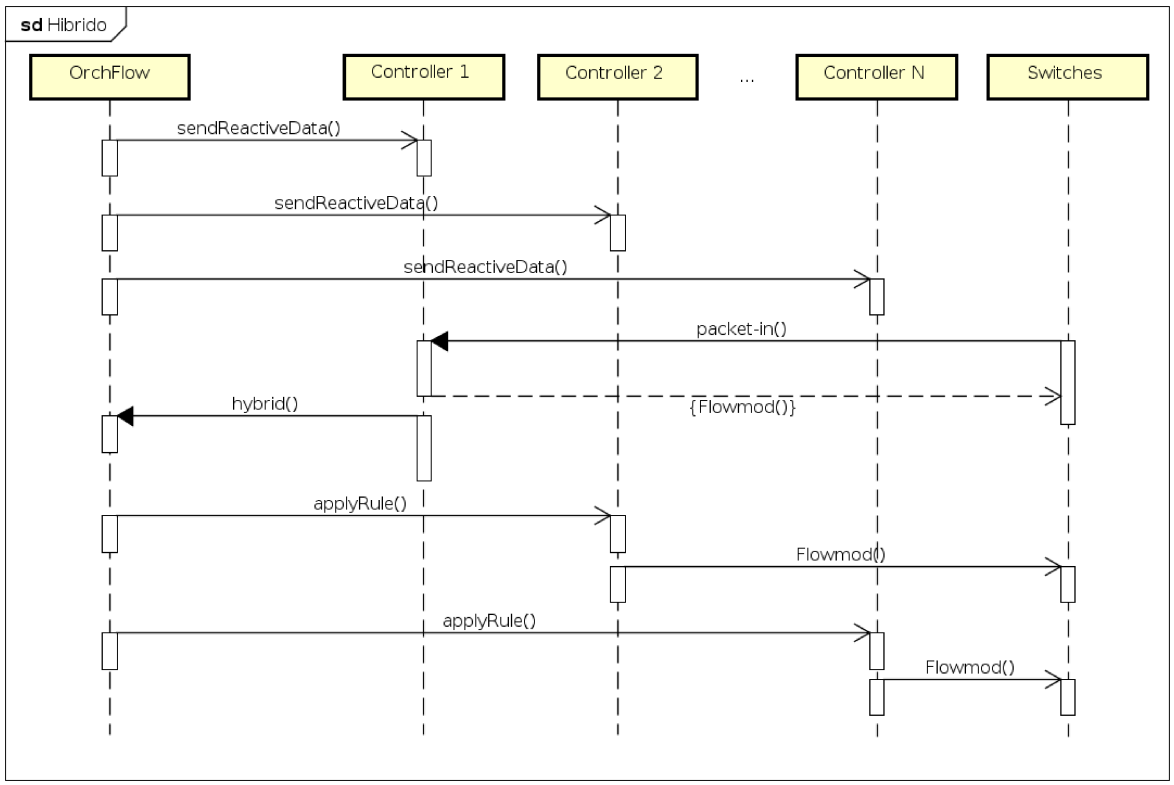


Figura 9 – Diagrama de Sequência - Modo Híbrido.

Fonte: Elaborada pelo autor

Assim, para todo e qualquer fluxo que chegue à rede, é necessário que haja uma regra correspondente na tabela de fluxos do primeiro *switch* (modo proativo). Entretanto, caso não haja uma regra correspondente, o *switch* encaminhará um evento *packet-in* ao controlador de subdomínio que buscará em sua tabela *Reactive* uma regra compatível com o serviço solicitado. Neste caso, os controladores devem possuir uma programação prévia para determinarem a configuração a todos os *switches* envolvidos neste serviço (modo reativo ou modo híbrido). Porém, caso um fluxo novo chegue à rede e o mesmo não encontre correspondência na tabela de fluxos ou não haja uma programação prévia que possa tratá-lo de forma correta, este será descartado, como é feito tradicionalmente em redes SDN.

3.2 Implementação

O OrchFlow foi desenvolvido em linguagem Java e utiliza um banco de dados não relacional, o Neo4j (EIFREM, 2009). O desenvolvimento do OrchFlow busca proporcionar a integração, a gerência, a administração e a programação para as redes SDN baseada em múltiplos controladores OpenFlow. Para isso ele deve ser capaz de ler e gravar as diversas programações em cada um dos controladores, de forma a garantir o funcionamento dos serviços entre quaisquer *end-hosts* pertencentes ao mesmo domínio administrativo, mesmo

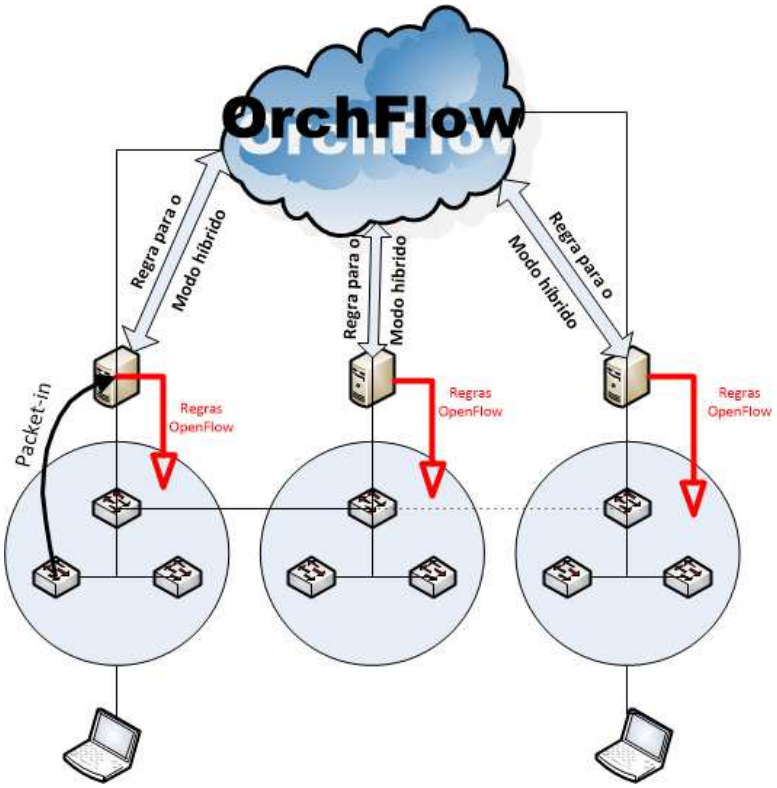


Figura 10 – Modo Hibrido.

Fonte: Elaborada pelo autor

estando localizados em diferentes subdomínios.

Para esta dissertação, cada subdomínio utiliza um controlador Floodlight ou Ryu, entretanto o uso de outros controladores já está em andamento e deverá ser explorado em trabalhos futuros.

3.2.1 Banco de Dados - Neo4j

Para que haja a correta leitura e consolidação dos dados é necessário obtê-los e armazená-los de forma eficiente e bem estruturada. Assim, foi escolhido o Neo4j, um banco de dados orientado a grafos baseado em Java que oferece persistência, alto desempenho, escalabilidade e possui uma comunidade ativa e uma boa documentação. Assim, o Neo4j recebe os dados de cada um dos *end-hosts* e *switches* e representa-os como nós, obtém os dados dos *links* e cria um relacionamento entre eles, possibilitando ao administrador obter uma visão completa de todo o domínio administrativo, através de uma única interface gráfica.

Para que o sistema funcione é necessário o reconhecimento de toda a topologia da rede. O OrchFlow obtém estes dados de cada controlador através da API REST já definida em cada controlador. Note que a maioria, se não todos os controladores, possuem uma chamada em REST para obtenção da topologia. Sendo assim, o OrchFlow se aproveita

desta interface já existente para obter de cada controlador a topologia de seu subdomínio e construir a topologia completa do domínio administrativo armazenando-a no banco de dados Neo4J.

Uma vez que se tenha armazenado todos os dados relacionados à rede, pode-se obter o melhor caminho entre dois *end-hosts* e implementar o serviço de roteamento fim-a-fim. Nesta pesquisa, todas as rotas definidas por solicitação de alguma aplicação ao OrchFlow são criadas tomando por base o algoritmo de Dijkstra (DIJKSTRA, 1959), que faz uma busca pelo caminho mais curto e determina aos controladores que criem os fluxos conforme o resultado encontrado. Assim, qualquer rota, mesmo as que passam por dois ou mais subdomínios, são configuradas por seus controladores conforme o que for definido pelo OrchFlow.

3.2.2 Módulos ARPReply e Reactive

Para identificar o correto funcionamento do OrchFlow, foi necessário desabilitar o módulo *Forward* de todos os controladores de forma que não interfiram no funcionamento da rede. Para todo e qualquer fluxo na rede, deverá existir uma regra criada e orquestrada pelo OrchFlow. Assim, faz-se necessário a criação de mecanismos de entrega de pacotes ARP em uma rede Ethernet, possibilitando que um *host* obtenha o endereço físico de um *host* destino quando ele necessitar enviar um pacote conhecendo apenas o endereço IP. Como mostrado na Figura 11, a abordagem do modelo tradicional para o tratamento do pacote *ARP Request* ocorre através do envio das mensagens em *broadcast*. Nesse modelo, quando um *host* não conhece o endereço físico do *host* destino, uma mensagem do tipo *broadcast* será encaminhada e todos os *hosts* no mesmo domínio de *broadcast* receberão a mensagem.

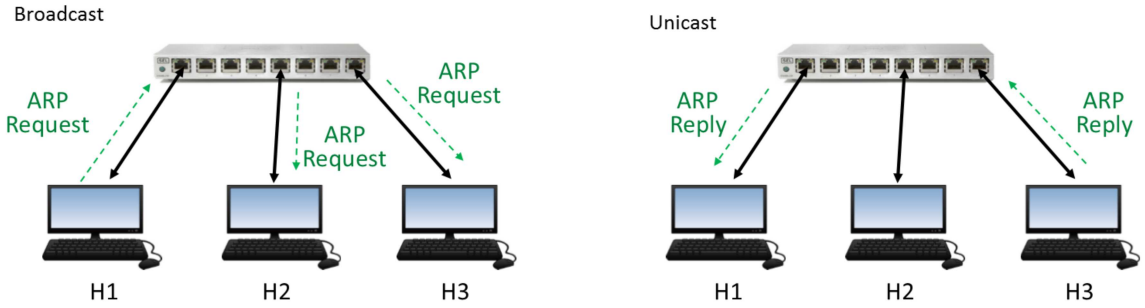


Figura 11 – Modelo tradicional para resolução do ARP.

Fonte: Elaborada pelo autor

Em (OLIVEIRA; VERDI, 2012), os autores comparam dois métodos de tratamento de mensagens *ARP Request* e conseguem melhorar o tempo de resposta da resolução em

uma rede SDN. Baseado neste trabalho, desenvolvemos um módulo capaz de responder às mensagens ARP dentro de cada subdomínio. Este módulo visa a criação de um pacote *ARP REPLY* para que a requisição *ARP REQUEST* seja respondida diretamente pelo controlador do subdomínio. Nela, o controlador usa todas as estruturas que possui para montar a resposta de um pacote ARP e enviá-la diretamente ao *host* solicitante.

Nesse sentido, cada controlador reconhece a topologia de seu subdomínio e a repassa ao OrchFlow. De posse das informações de todo o domínio administrativo, cria-se então uma tabela de conversão MAC/IP. Tal tabela é enviada para todos os controladores através de uma interface REST estendida. O módulo ARPReply previamente instalado irá armazenar esta tabela para montar respostas às requisições ARP. Assim, mesmo que um *host* solicite a resolução ARP de um *host* localizado em outro subdomínio, o controlador já terá os dados necessários para devolução da correta mensagem de retorno do protocolo ARP. Como mostrado na Figura 12, ao chegar um pacote *ARP REQUEST*, o *switch* encaminha um evento *Packet-in* para o controlador, que após pesquisar em sua tabela Reactive, encaminhará de volta um *Packet-out* para o *switch* com o *ARP Reply* correspondente. Essa solução traz um ganho à rede, eliminando assim, o *broadcast* de mensagens ARP.

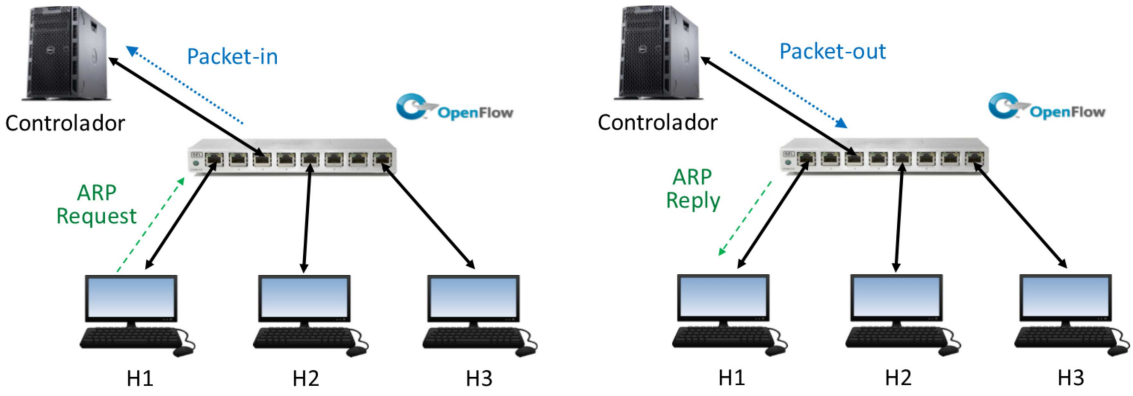


Figura 12 – Eliminando o broadcast através do módulo ARPReply.

Fonte: Elaborada pelo autor

Além do ARPReply, também desenvolvemos o módulo *Reactive* responsável por receber as regras do OrchFlow e armazená-las em uma estrutura interna. Estas regras são posteriormente utilizadas para tratamento de eventos de *packet-in* originados na rede. Este modo de ação, chamado aqui de modo reativo, permite que se use as tabelas de fluxos de maneira mais eficaz diminuindo as entradas na memória TCAM de cada *switch* na rede.

3.2.3 Interface

A ferramenta apresenta uma interface simples onde os serviços a serem aprovisionados no domínio administrativo podem ser solicitados. Inicialmente, como mostrado na Figura 13, os controladores devem ser cadastrados, inserindo opcionalmente um nome e de forma obrigatória o endereço IP da interface web e porta http.

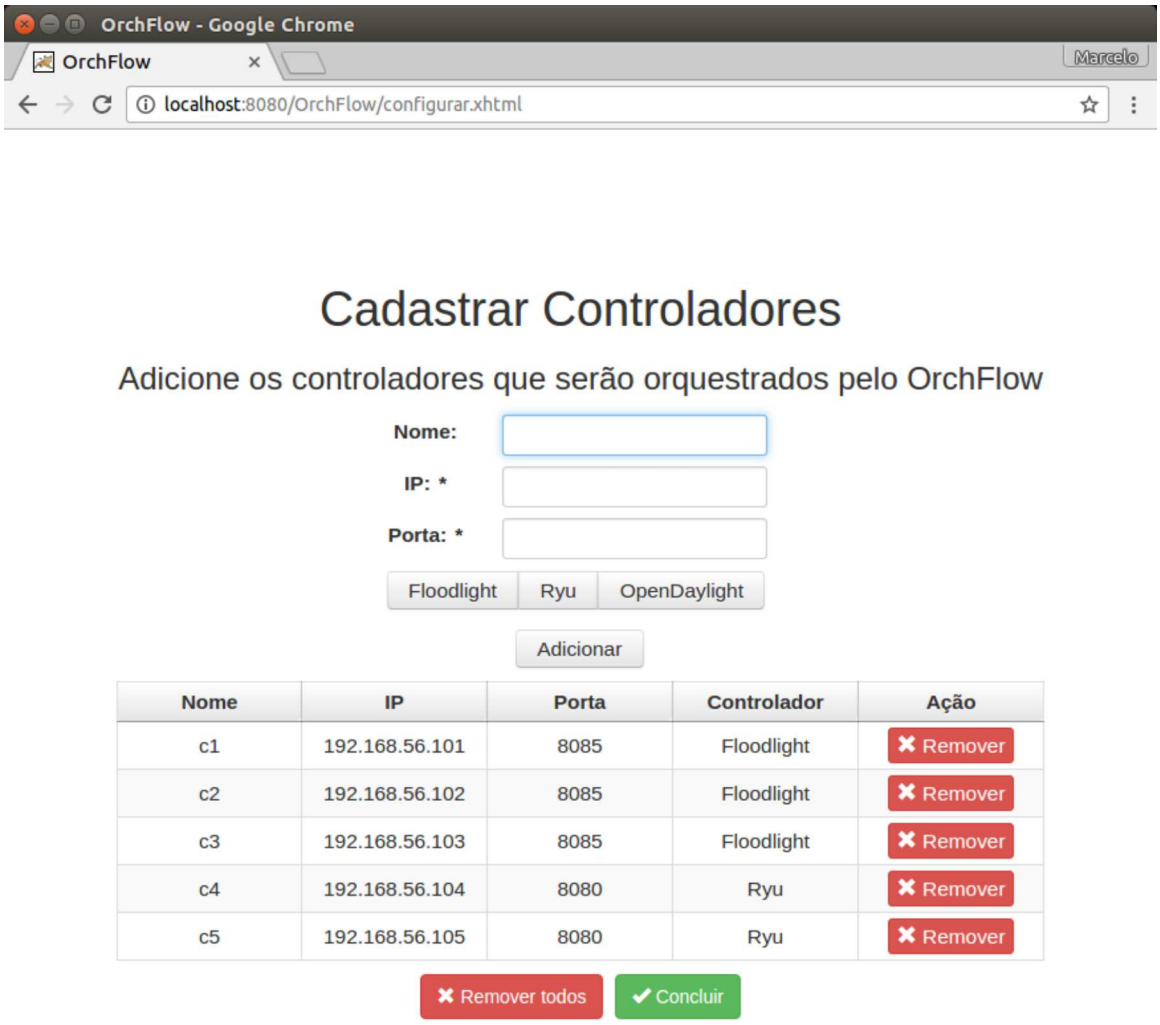


Figura 13 – Tela de cadastro dos controladores.

Fonte: Elaborada pelo autor

Observe aqui, que para o funcionamento correto é preciso indicar a implementação do controlador, pois no atual estágio de desenvolvimento do OrchFlow ainda não é possível identificar de forma automática os controladores. Este procedimento é obrigatório, pois só assim o OrchFlow poderá realizar as consultas e determinar as configurações a cada subdomínio.

Uma vez cadastrados todos os controladores, o sistema estará pronto para uso, carregando a página conforme a Figura 14. Nesta página, é possível analisar a topologia da rede, alterando cores, mudando posições de nós e *links* dentre outras opções.

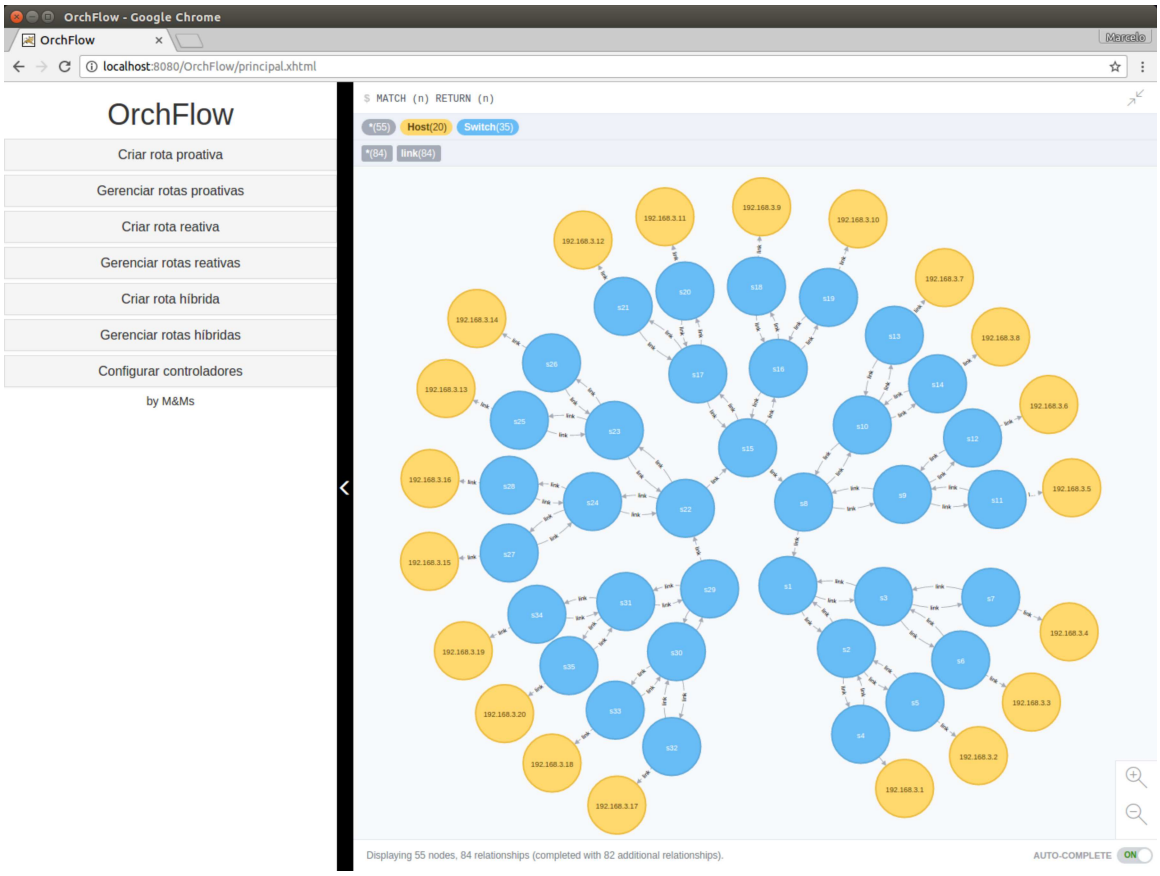


Figura 14 – Interface do OrchFlow.

Fonte: Elaborada pelo autor

Do lado esquerdo da interface é possível identificar os três modos de operação do OrchFlow no menu de opções: modos proativo, reativo e híbrido. É possível observar também que o usuário poderá gerenciar os serviços para cada um dos três modos de operação. Para todos os modos o procedimento para solicitar o serviço de roteamento fim-a-fim é o mesmo. Nos três casos, ao clicar no menu "criar rota proativa, reativa ou híbrida", o usuário terá à sua disposição os campos do protocolo OpenFlow para determinar as características de cada fluxo desejado. Como mostrado na Figura 15, é preciso escolher os *hosts* de origem e destino, criar um nome para a rota e definir os parâmetros seguintes conforme o serviço solicitado.

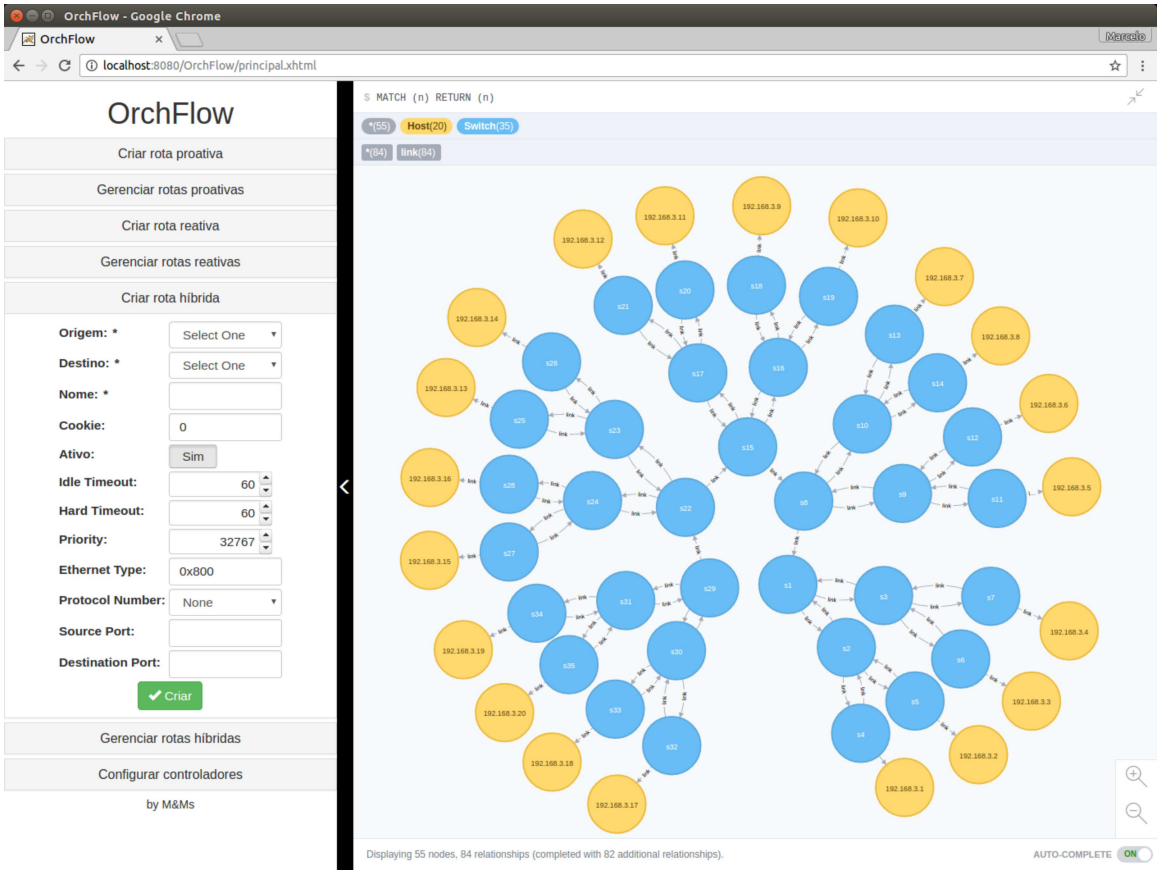


Figura 15 – Menu - Modo Híbrido.

Fonte: Elaborada pelo autor

4 Resultados e Análise

Neste capítulo, são apresentados os resultados dos testes realizados com os protocolos ICMP, UDP e TCP.

4.1 Ambiente de testes

Para a validação da solução proposta foi utilizado um computador com processador: Intel® Core™ i7-4510U CPU 2.00GHz, com 16GB de memória RAM e sistema operacional Ubuntu 14.04 LTS. A topologia adotada para esta dissertação é formada por um domínio administrativo subdividido em cinco subdomínios, com duas implementações diferentes de controladores, o Floodlight e o Ryu[1]. Cada controlador é responsável pelo controle e gerenciamento de um único subdomínio e cabe ao OrchFlow a orquestração, gerencia e a visão completa do domínio administrativo. As cinco redes estão definidas e interligadas através de *links* específicos, denominados aqui de *links* externos, utilizando ainda *switches* virtuais *OVS-Switch* e o protocolo OpenFlow na versão 1.3.

Como se pode ver na Figura 16, toda a implementação da topologia utilizada nos testes do OrchFlow foi realizada através de seis máquinas virtuais (VMs) configuradas da seguinte forma: uma CPU, 512MB de memória RAM e Sistema Operacional Ubuntu Server 12.04:

VM1: Nesta VM uma rede é emulada através do emulador Mininet (LANTZ; HELLER; MCKEOWN, 2010), contendo cinco subdomínios interligados em barramento, com 7 *switches* em forma de árvore e 4 *hosts* cada.

VM2, VM3, VM4, VM5 e VM6: Nas três primeiras VMs tem-se um controlador Floodlight e nas outras duas um controlador Ryu, funcionando e prontos para receber os eventos vindos dos *switches* das redes emuladas na VM1.

Em todos os testes, a mesma topologia foi utilizada e foram criadas rotas fim-a-fim entre um *host* do primeiro subdomínio e os *hosts* dos demais subdomínios para os três modos de atuação: proativo, reativo e híbrido.

Nesta dissertação, o protocolo ICMP permite testar a conectividade e identificar a quantidade de pacotes perdidos em decorrência do tempo necessário para o estabelecimento da rota fim-a-fim entre dois *hosts*. De maneira similar, o protocolo UDP permite verificar a quantidade de datagramas perdidos, até que se estabeleça a comunicação entre os *hosts*. Por fim, o protocolo TCP permite determinar o tempo total necessário para transmitir uma determinada quantidade de bytes entre dois *hosts*, o *Flow-Completion Time* (FCT).

[1] https://osrg.github.io/ryu/

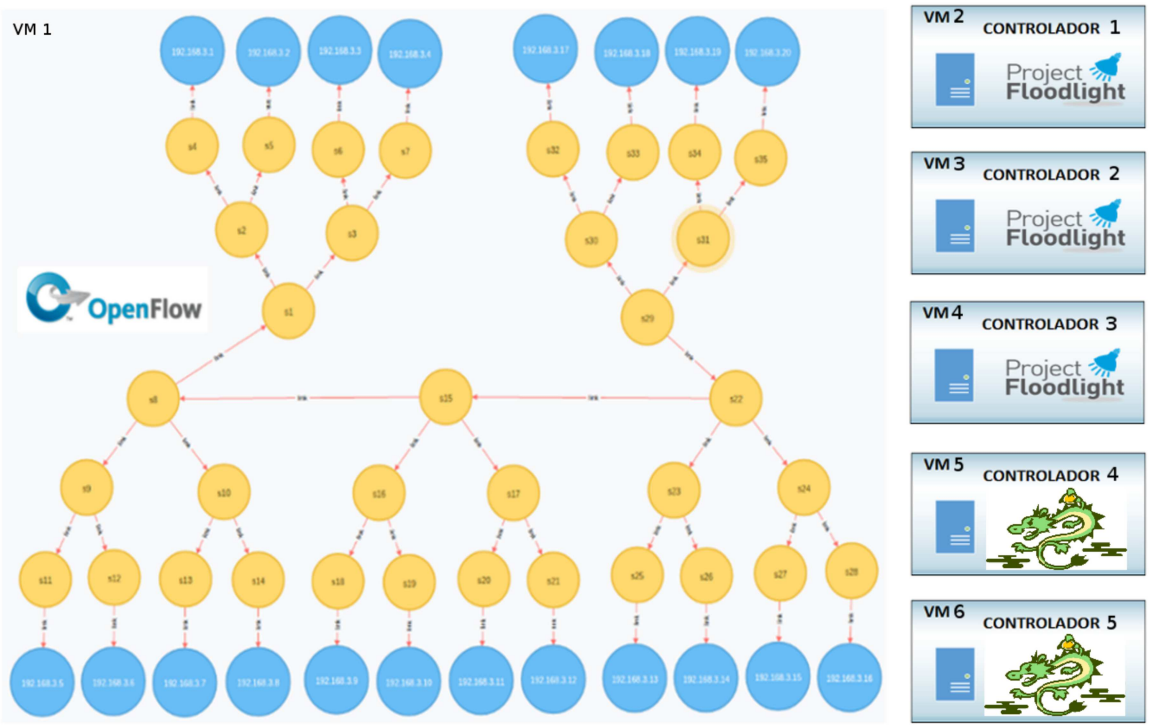


Figura 16 – Topologia de rede.

Fonte: Elaborada pelo autor

Para a realização dos testes com o protocolo ICMP, foi utilizado o comando ping no primeiro *host*, e observado a quantidade de pacotes perdidos durante a comunicação. Como mostrado no exemplo da Figura 17, este comando retorna ao término de cada teste a porcentagem de pacotes perdidos na comunicação fim-a-fim. Neste caso, um ping do *host* h1 para o h20, gerou 50% de perda dos pacotes enviados.

Para os protocolos UDP e TCP, fez-se necessário a utilização de uma ferramenta de medição e geração de tráfego de dados do tipo cliente/servidor. Assim, a ferramenta Iperf (TIRUMALA et al., 2005), desenvolvida com código livre e gratuita, foi escolhida para medir as perdas de datagramas e o FCT.

Para estes testes são necessários a execução de um comando iperf em cada *host*. Deste modo, um teste com o protocolo UDP pode ser visto no exemplo mostrado através da Figura 18, onde um servidor é iniciado no *host* h16, Figura 18a e fica aguardando a comunicação originada pelo cliente h1, mostrado na Figura 18b.

O comando executado no *host* cliente deu inicio a transferência de 11,9 MBytes a uma taxa de 10,0 Mbits/segundo para o *host* servidor. No entanto, neste exemplo, ocorreu a perda de 40 de um total de 890 datagramas enviados no primeiro intervalo de um segundo, totalizando 4,49% de perdas neste primeiro segundo de transmissão.

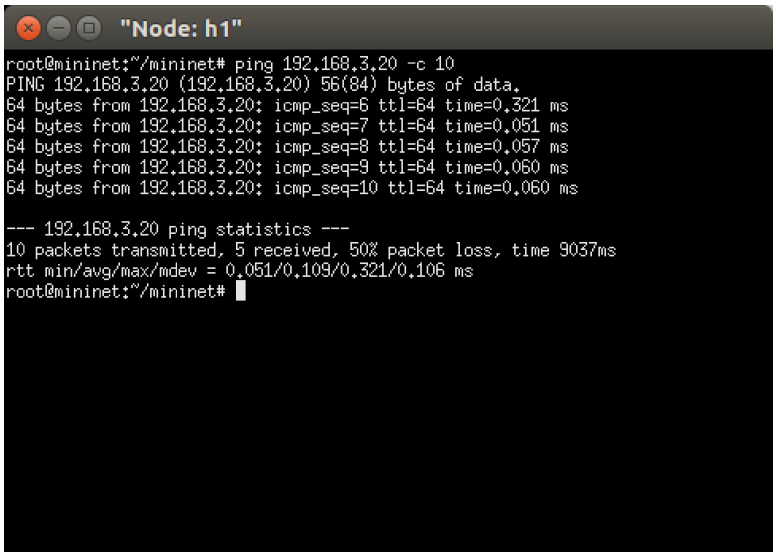
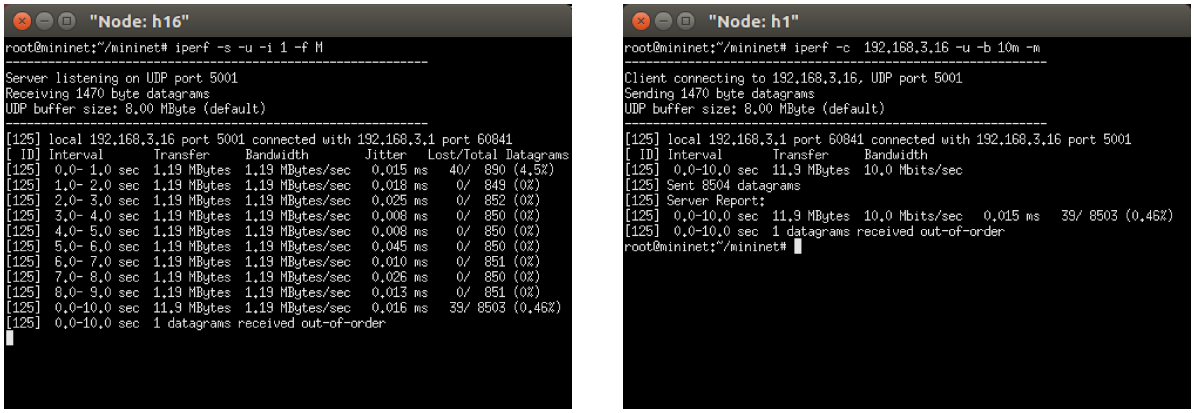


Figura 17 – Comando ping.

Fonte: Elaborada pelo autor



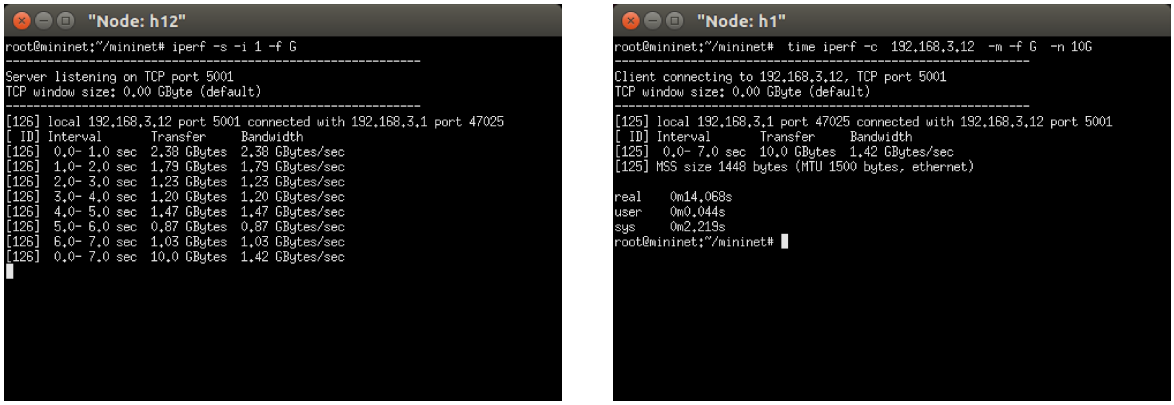
(a) Servidor (b) Cliente

Figura 18 – Protocolo UDP.

Fonte: Elaborada pelo autor

Para o protocolo TCP, Figura 19, o servidor iperf é executado conforme a Figura 19a no *host* h12. O cliente foi executado no *host* h1, como mostrado pela Figura 19b. Neste exemplo, o *host* cliente gera um tráfego de 10 GBytes, fazendo o máximo uso da largura de banda disponível. Aqui a taxa média de transferência foi de 1,42 Gbytes/segundo.

Apesar do iperf mostrar que a transferência ocorre em sete segundos, é preciso saber qual foi o tempo total para a conclusão do teste. Assim, foi necessário utilizar o comando *time* do Linux, onde foi possível verificar a soma dos tempos utilizados para a transferência do conjunto de Bytes gerado pela ferramenta iperf, acrescido do tempo necessário para o estabelecimento da rota fim-a-fim entre os *hosts* h1 e h12, passando por todos os *switches* e controladores do domínio administrativo.



(a) Servidor (b) Cliente

Figura 19 – Protocolo TCP.

Fonte: Elaborada pelo autor

Neste exemplo o comando time associado ao comando iperf, possibilitaram verificar que o FCT para transferir os 10 GBytes do *host* h1 até o h12, passando por nove *switches* em cinco subdomínios diferentes foi de 14,07 segundos.

4.2 Resultados

Para a validação dos diferentes cenários, todos os testes foram realizados trinta vezes, obtendo-se a média dos valores apresentados para cada rota criada e em cada modo de operação para cada um dos três protocolos utilizados.

Para todos os modos de operação, foram realizados os testes em três etapas: a primeira utilizou apenas a implementação do controlador Ryu em cinco subdomínios, na segunda apenas a implementação do controlador Floodlight em dez subdomínios, e na terceira foram utilizadas as duas implementações. Neste último caso, em dois cenários diferentes: o primeiro com três controladores Floodlight e dois controladores Ryu e o segundo com dois controladores Floodlight e três controladores Ryu.

Nesta dissertação, os testes visam comprovar o funcionamento do OrchFlow em relação aos diversos tipos de implementações de controladores existentes, possibilitando avaliar o funcionamento de controladores desenvolvidos em linguagens de programação diferentes. Os controladores Ryu escritos em Python e os controladores Floodlight escritos em Java, serão demonstrados a seguir. Vale observar, que a utilização de outras implementações de controladores, podem ser feitas através do desenvolvimento dos módulos de integração ao OrchFlow.

4.2.1 Controlador Ryu

Nesta seção, são apresentados os resultados dos testes realizados nos modos proativo, reativo e híbrido, utilizando apenas a implementação do controlador Ryu.

4.2.1.1 Proativo

Como explicado no Capítulo 3.1.1, no modo proativo, após uma aplicação solicitar o estabelecimento de uma rota fim-a-fim, toda a programação é previamente executada pelo OrchFlow e transferida aos controladores. Estes aplicam todas as regras diretamente nas tabelas de fluxos dos *switches* envolvidos. Assim, quando um novo fluxo de dados chegar na rede OpenFlow, será encaminhado diretamente sem gerar um evento *packet-in*. Este modo oferece aos fluxos o encaminhamento imediato, sem perdas ou prejuízo com relação ao FCT para a aplicação solicitante.

No modo proativo, foram realizados aproximadamente 120 repetições dos testes com os protocolos ICMP e UDP, utilizando o comando ping e a ferramenta Iperf, entre *hosts* do primeiro subdomínio e os *hosts* dos demais subdomínios. Como esperado, não ocorreram perdas em nenhum dos testes realizados, demonstrando assim, o correto funcionamento da implementação em todos os cenários utilizados.

Para melhor análise dos resultados encontrados e compreensão do funcionamento da arquitetura OrchFlow, foram realizados testes utilizando o protocolo TCP, onde pode-se observar o FCT com o tempo total necessário para transmitir uma determinada quantidade de bytes entre dois *hosts*. Neste caso, os tempos de FCT para o controlador Ryu, iniciaram em 5,00 segundos para o primeiro subdomínio e chegaram a 5,76 segundos no quinto subdomínio. Como mostrado na Figura 20, esses tempos vão crescendo proporcionalmente à quantidade de *switches* e subdomínios existentes entre os *hosts* envolvidos na comunicação fim-a-fim.

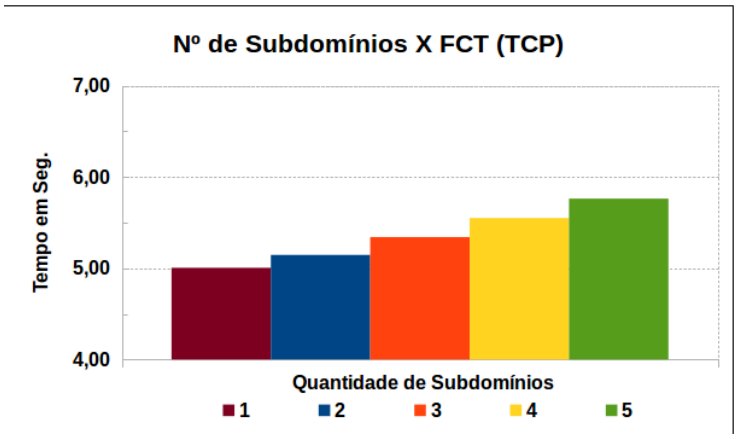


Figura 20 – Modo Proativo - Ryu - TCP.

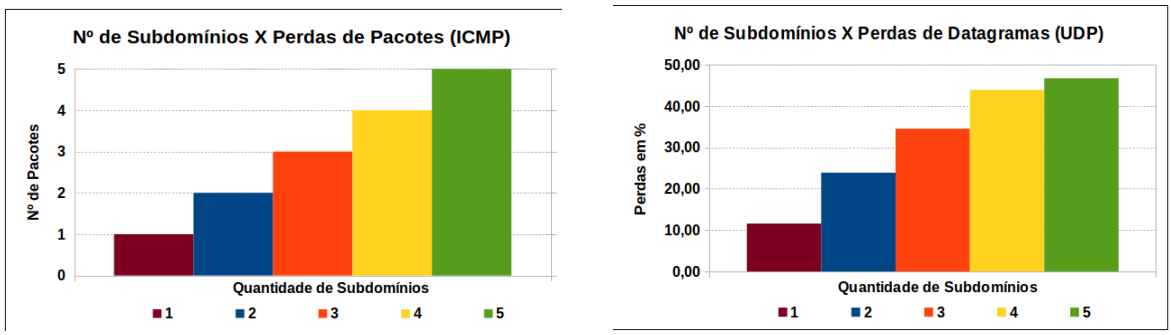
Fonte: Elaborada pelo autor

Pode-se perceber que a medida em que a quantidade de *switches* e subdomínios aumentam, aumenta também os tempos obtidos de FCT.

4.2.1.2 Reativo

No modo reativo, o comando ping foi utilizado para testar a conectividade entre um *host* do primeiro subdomínio e os demais *hosts* de todos os subdomínios. Isto permitiu verificar se as rotas entre os diversos subdomínios foram criadas corretamente. A Figura 21a mostra a quantidade de pacotes perdidos por subdomínios, evidenciando o crescimento conforme cresce também a quantidade de subdomínios envolvidos na rota fim-a-fim, perdendo-se um pacote a cada subdomínio por onde o fluxo passa.

Conforme mostrado na Figura 21b, sempre que um *host* inicia uma comunicação na rede, ocorrem perdas de datagramas até que o estabelecimento da rota fim-a-fim se conclua. É possível perceber que a quantidade de datagramas perdidos na transmissão entre dois *hosts*, do primeiro subdomínio, é significativamente menor que as perdas entre um *host* do primeiro subdomínio e outro do quinto subdomínio. É possível observar também que tais perdas crescem na medida em que cresce a quantidade de subdomínios envolvidos na comunicação. Nota-se que a cada novo subdomínio aumentam as perdas de datagramas.



(a) ICMP - PING (b) UDP

Figura 21 – Modo Reativo - Ryu.

Fonte: Elaborada pelo autor

A Figura 22 mostra os tempos em segundos relativos ao FCT usando o protocolo TCP no modo reativo. É possível perceber que este tempo aumenta na medida em que cresce a quantidade de subdomínios. Cabe observar aqui que o evento *Packet-in* ocorre sempre através do primeiro *switch* em cada subdomínio. Assim, caso um fluxo trafegue de um subdomínio para outro, um novo evento *Packet-in* é gerado no primeiro *switch* de cada subdomínio até que o fluxo chegue ao seu *host* de destino.

No modo reativo, o primeiro pacote do fluxo faz com que o *switch* dispare o evento *packet-in* ao controlador e este responderá com uma regra pré-estabelecida pela aplicação solicitante, implantando-as em todos os *switches* envolvidos na comunicação fim-a-fim

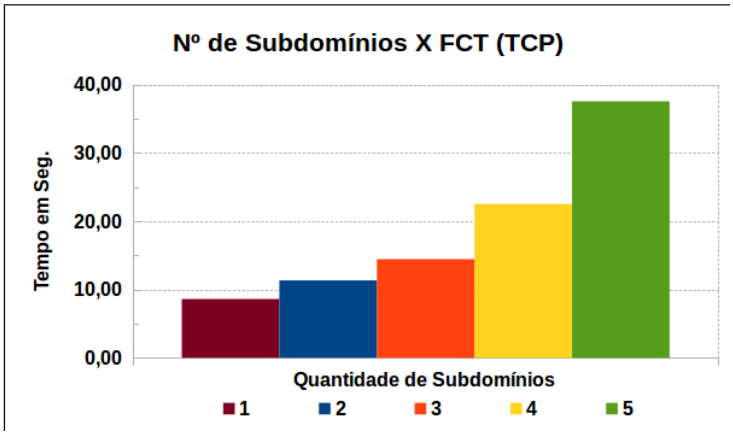


Figura 22 – Modo Reativo - Ryu - TCP.

Fonte: Elaborada pelo autor

de seu subdomínio. Entretanto, cada novo fluxo traz um atraso para sua configuração, aumentando o fluxo de mensagens entre o controlador e os diversos *switches* de seu subdomínio, fazendo com que cresça o número de datagramas perdidos no início de uma transmissão UDP ou que se tenha perda de tempo em uma transmissão TCP.

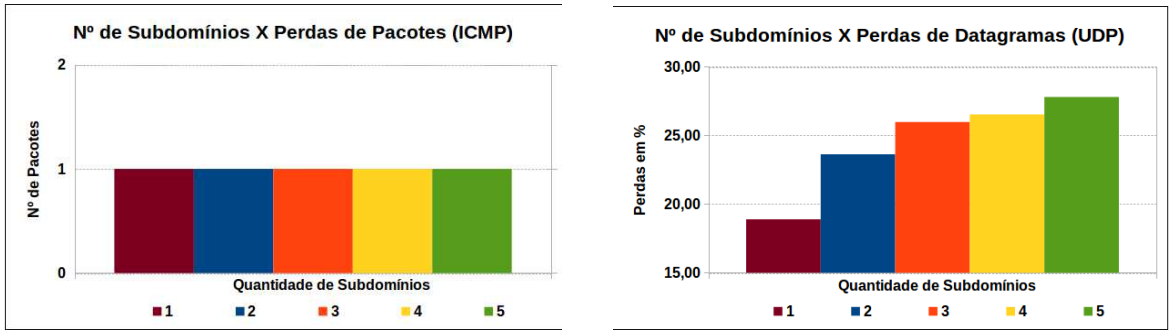
4.2.1.3 Híbrido

O OrchFlow, através do modo híbrido, permitirá ao administrador de redes programar todo o seu domínio administrativo de maneira idêntica ao modo reativo, dispondo das vantagens do modo proativo. Este mecanismo atua inicialmente de maneira reativa mas, após o primeiro evento de *Packet-in* no primeiro subdomínio, atua de maneira proativa nos demais subdomínios. No melhor caso, nenhum outro evento de *Packet-in* será disparado nos próximos subdomínios, diminuindo assim as perdas e o tempo de transferência total dos dados. Já no pior caso, o modo híbrido se comportará de maneira idêntica ao modo Reativo.

Para este modo, os testes foram realizados seguindo os mesmos parâmetros que o modo reativo. Como mostrado na Figura 23a, através do comando ping, as perdas ocorrem apenas no primeiro subdomínio. Este modo torna, portanto, o tráfego do fluxo mais rápido nos demais subdomínios.

Para o protocolo UDP, obteve-se a quantidade de datagramas perdidos, e como mostrado na Figura 23b, as perdas iniciam em 19,97% e continuaram a crescer. Tais perdas ocorrem devido ao tempo de processamento do OrchFlow e à comunicação via interface HTTP entre o OrchFlow e os controladores dos diversos subdomínios.

Observa-se aqui, que ocorre inicialmente maior perda de datagramas comparados ao modo reativo. No entanto, elas crescem até um certo limite, estabilizando-se conforme cresce o número de subdomínios, mantendo nesse caso, um número máximo na ordem de



(a) ICMP - PING (b) UDP

Figura 23 – Modo Híbrido - Ryu.

Fonte: Elaborada pelo autor

27,41% das perdas.

Para comprovar a funcionalidade do OrchFlow no modo híbrido, foram realizados os testes utilizando o protocolo TCP e, como mostrado na Figura 24, o FCT para a transmissão dos 10 GBytes foi em média 8,00 segundos. Iniciando em 7,09 segundos para o primeiro subdomínio indo até 8,76 segundos no quinto subdomínio.

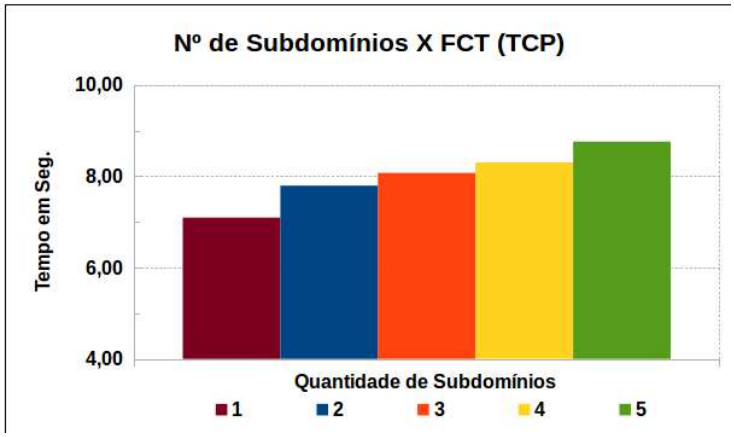


Figura 24 – Modo Híbrido - Ryu - TCP.

Fonte: Elaborada pelo autor

Percebe-se que o FCT para o primeiro subdomínio nos modos reativo e híbrido ficou muito próximo e que nos demais subdomínios os valores aumentam de forma semelhante ao modo proativo.

4.2.2 Controlador Floodlight

Nesta seção são apresentados os resultados com os testes realizados nos modos proativo, reativo e híbrido utilizando apenas a implementação do controlador Floodlight. Pensando em diminuir as perdas encontradas nos testes realizados com o controlador

Ryu no primeiro subdomínio, buscou-se melhorar o funcionamento do módulo Reactive, incluindo o processamento paralelo através dos diferentes controladores, usando *Threads*.

Para a demonstração do correto funcionamento dessa implementação, foi necessário o uso de uma nova topologia de rede, capaz de mostrar a estabilização das perdas no modo híbrido. A topologia adotada desta vez, envolve dez subdomínios com as mesmas configurações adotadas na Seção 4.2.1. Cada subdomínio é composto por sete *switches* em forma de árvore e quatro *hosts*.

4.2.2.1 Proativo

No modo proativo, os tempos de FCT para o controlador Floodlight iniciaram em 5,60 segundos para o primeiro subdomínio e chegaram a 6,05 segundos no décimo subdomínio. Como mostrado na Figura 25, esses tempos vão crescendo conforme cresce a quantidade de *switches* e subdomínios existentes entre os *hosts* envolvidos na comunicação fim-a-fim.

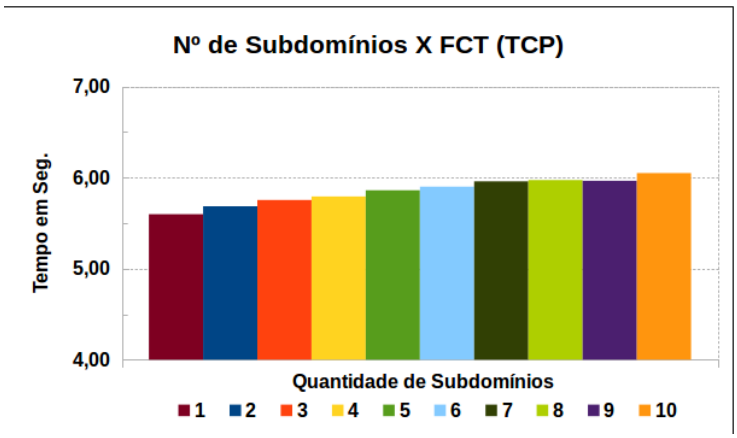


Figura 25 – Modo Proativo - Floodlight - TCP.

Fonte: Elaborada pelo autor

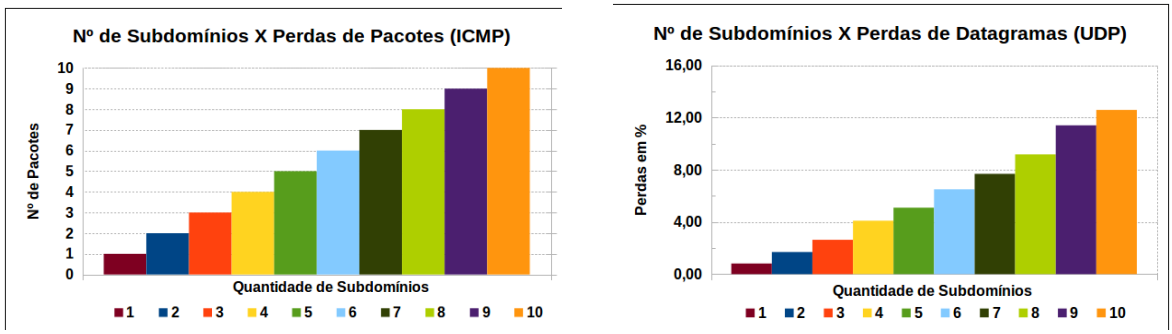
Neste modo, as duas implementações apresentam resultados semelhantes, pois as regras são pré-estabelecidas nos *switches*, tornando o seu funcionamento livre da interferência dos controladores, permitindo a passagem dos fluxos sem a necessidade de comunicação com os controladores.

4.2.2.2 Reativo

No modo reativo, as perdas em relação ao uso do protocolo ICMP foram equivalentes às apresentadas com o controlador Ryu na Seção 4.2.1.2, e como mostrado na Figura 26a, perde-se um pacote a cada novo subdomínio.

No entanto, ao utilizar o protocolo UDP no Floodlight, percebe-se que a quantidade de datagramas perdidos foi reduzida em relação aos valores obtidos com os controladores

Ryu e, conforme mostrado na Figura 26b, as perdas encontradas no primeiro subdomínio chegam a menos de 1%.



(a) ICMP - PING (b) UDP

Figura 26 – Modo Reativo - Floodlight.

Fonte: Elaborada pelo autor

Programas escritos em Python, em algumas situações, podem ser mais lentos para executar, e como se percebe aqui, uma linguagem interpretada durante a execução torna o controlador Ryu mais lento que o controlador Floodlight, que é escrito em Java e compilado. Além disso, o uso de técnicas de processamento paralelo possibilitaram a queda na quantidade de datagramas perdidos no primeiro segundo de transmissão usando o controlador Floodlight. Entretanto, o funcionamento do OrchFlow continua o mesmo, ou seja, as perdas crescem na medida em que cresce também a quantidade de subdomínios envolvidos na comunicação.

Para o protocolo TCP, é possível perceber através da Figura 27, que os tempos de reação dos controladores Floodlight são praticamente iguais aos controladores Ryu. Entretanto, a partir do sétimo subdomínio, o FCT ultrapassa o tempo de dois minutos, resultando em falha de conexão. Por padrão, o iperf responde com uma mensagem de *timeout*, caso o servidor não responda em até dois minutos.

4.2.2.3 Híbrido

Seguindo os mesmos testes realizados e demonstrados na Seção 4.2.1.3, pode-se observar através da Figura 28a, que o modo híbrido está funcionando conforme o esperado. O protocolo ICMP, através do comando ping, demonstra as perdas ocorridas no primeiro subdomínio e que não ocorrem mais perdas para os demais subdomínios.

Ao utilizar o protocolo UDP, percebe-se as diferenças nas perdas ocorridas entre ambas as implementações de controladores. Pois, como mostrado na Figura 28b, as perdas iniciais foram muito inferiores às dos controladores Ryu. Entretanto, mesmo crescendo a cada subdomínio, tais perdas não se estabilizaram a tempo do fluxo chegar ao 5º subdomínio, estabilizando-se apenas à partir do oitavo subdomínio.

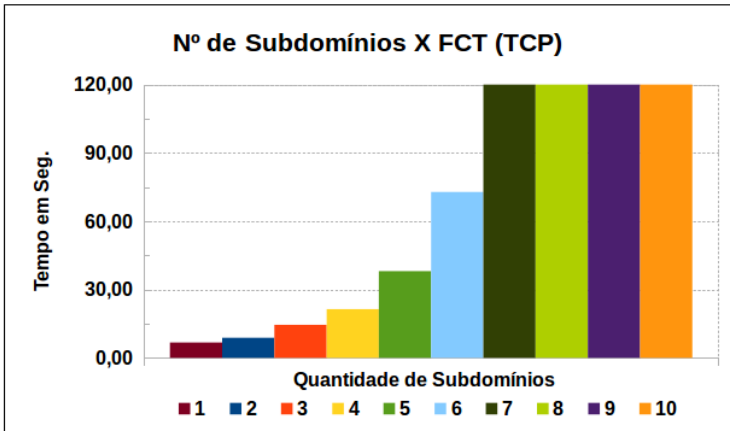
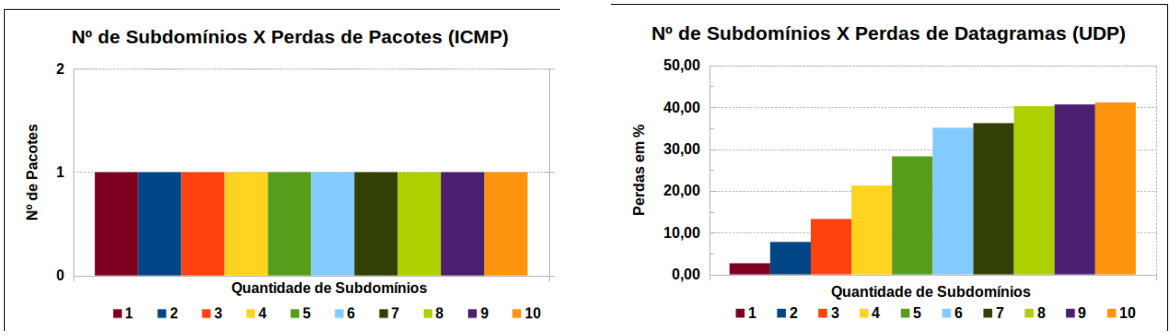


Figura 27 – Modo Reativo - Floodlight - TCP.

Fonte: Elaborada pelo autor



(a) ICMP - PING (b) UDP

Figura 28 – Modo Híbrido - Floodlight.

Fonte: Elaborada pelo autor

Para o protocolo TCP, no modo híbrido, utilizando os controladores Floodlight, os resultados foram próximos aos obtidos através dos controladores Ryu. E, conforme mostrado na Figura 29, os tempos para o FCT ficaram na média de 8,30 segundos.

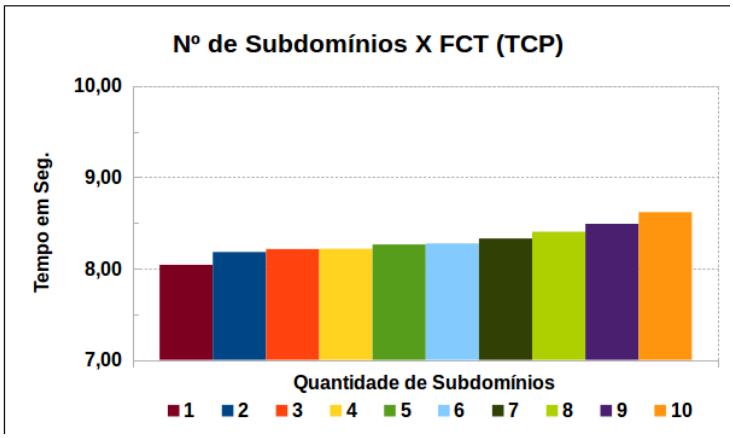


Figura 29 – Modo Híbrido - Floodlight - TCP.

Fonte: Elaborada pelo autor

Percebe-se então que as linguagens de programação utilizadas nos controladores podem interferir no funcionamento dos modos de operação do OrchFlow, diminuindo as perdas de datagramas no protocolo UDP. No entanto, o tempo de resposta para as aplicações que utilizam o protocolo TCP não apresentaram diferenças significativas.

4.2.3 Controladores Floodlight e Ryu

Nesta seção são apresentados os resultados dos testes realizados nos modos proativo, reativo e híbrido, utilizando conjuntamente as implementações dos controladores Ryu e Floodlight.

4.2.3.1 Três controladores Floodlight e dois controladores Ryu

Tendo sido testado as duas implementações de controladores em separado, e visto o seu funcionamento, foi necessário juntá-las numa única topologia e testar a capacidade que o OrchFlow tem de orquestrar múltiplos controladores com diferentes implementações. Desta forma, os testes foram realizados utilizando a mesma topologia de rede, com cinco subdomínios. No entanto, foram utilizados três controladores Floodlight e dois controladores Ryu, onde foi possível verificar a integração de implementações de controladores diferentes, orquestrados e trabalhando em conjunto sob a administração do OrchFlow, em um único domínio administrativo.

4.2.3.1.1 Proativo

No modo proativo, os tempos de FCT para este cenário iniciaram em 4,88 segundos para o primeiro subdomínio e chegaram a 5,47 segundos no quinto subdomínio, como mostrado na Figura 30.

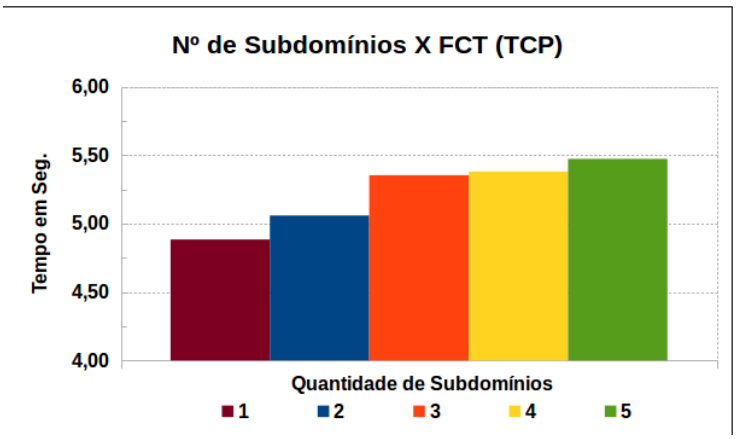
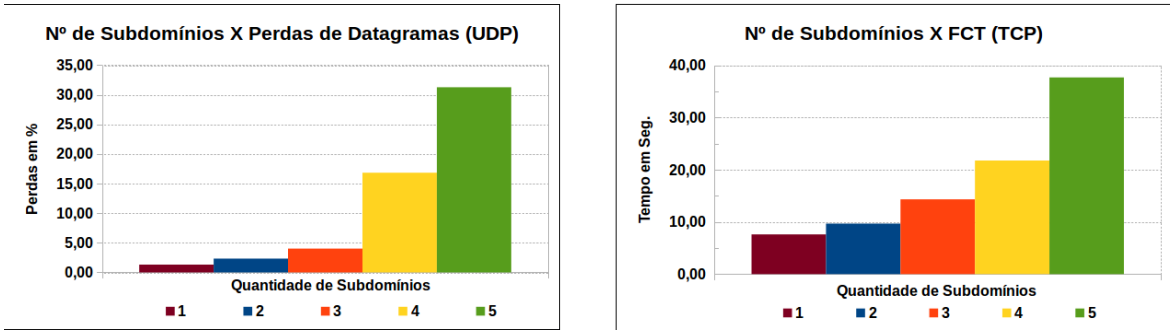


Figura 30 – Modo Proativo - 3 Floodlight + 2 Ryu - TCP.

Fonte: Elaborada pelo autor

4.2.3.1.2 Reativo

No modo reativo, é possível observar a influência das diferentes implementações. A Figura 31a demonstra os resultados obtidos, onde pode-se destacar que as perdas no modo reativo são muito baixas nos 3 controladores Floodlight e aumentam significativamente ao chegarem nos controladores Ryu, ultrapassando a casa dos 30%. No entanto, como mostrado na Figura 31b, os tempos para FCT continuam iguais aos vistos nos experimentos anteriores.



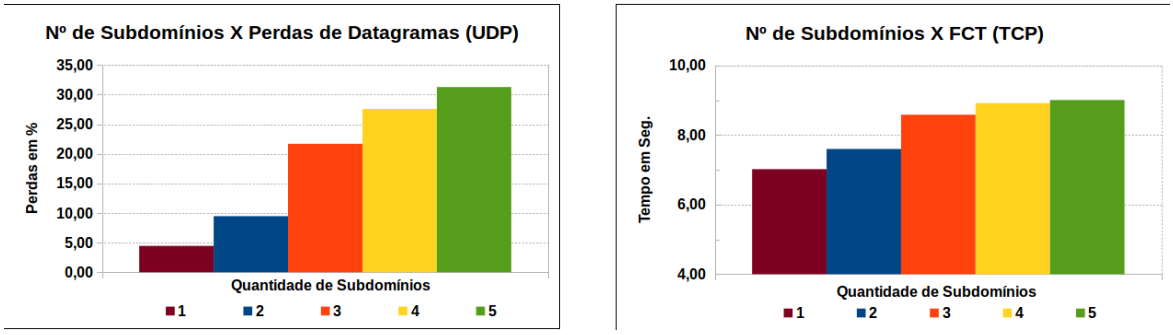
(a) UDP (b) TCP

Figura 31 – Modo Reativo - 3 Floodlight + 2 Ryu.

Fonte: Elaborada pelo autor

4.2.3.1.3 Híbrido

Para o modo híbrido, é possível perceber através da Figura 32a que neste cenário também não ocorreu a estabilização das perdas até o 5º subdomínio utilizando o protocolo UDP. Porém, como mostrado na Figura 32b, percebe-se aqui o funcionamento correto da arquitetura através do protocolo TCP, mantendo a média dos valores de FTC em 8,22 segundos.



(a) UDP (b) TCP

Figura 32 – Modo Reativo - 3 Floodlight + 2 Ryu.

Fonte: Elaborada pelo autor

4.2.4 Dois controladores Floodlight e três controladores Ryu

Buscando evidenciar o funcionamento do modo híbrido em um cenário com múltiplos controladores, foram refeitos os testes com uma nova configuração. Desta vez, utilizando-se 2 controladores Floodlight e 3 controladores Ryu. A Figura 33 demonstra o resultado das perdas com o protocolo UDP, obtidas através do modo reativo, onde percebe-se um aumento considerável a partir do 3º subdomínio, evidenciando as diferenças entre os controladores.

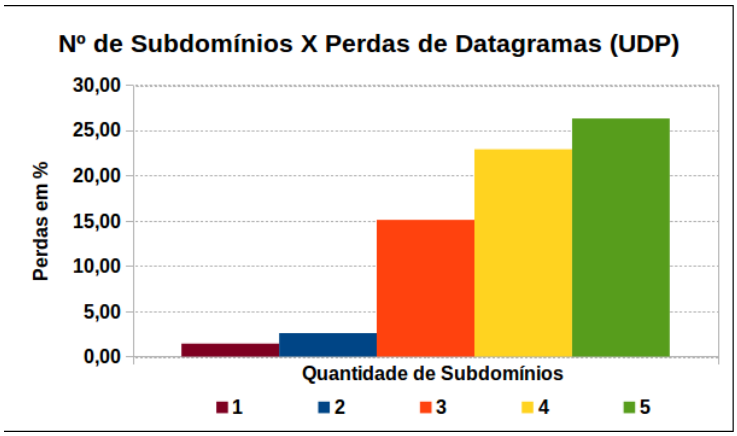


Figura 33 – Modo Reativo - 2 Floodlight + 3 Ryu - UDP.

Fonte: Elaborada pelo autor

Porém, é no modo híbrido que os resultados obtidos apresentam a maior diferença. Como mostrado na Figura 34, as perdas com o protocolo UDP aumentam consideravelmente no 3º subdomínio, seguindo de uma estabilização a partir do 3º subdomínio.

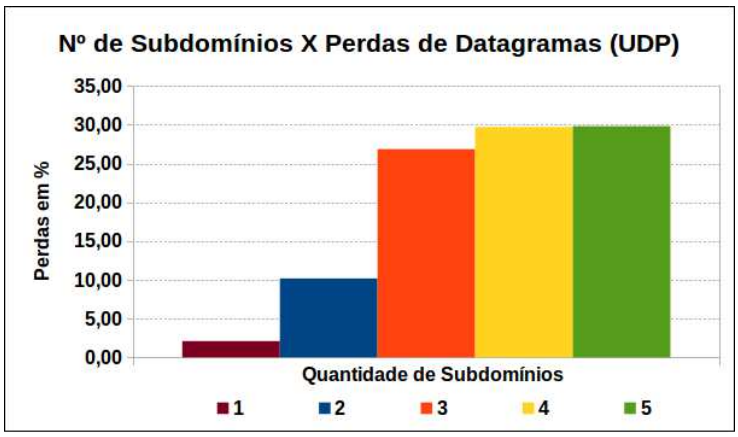


Figura 34 – Modo Híbrido - 2 Floodlight + 3 Ryu - UDP.

Fonte: Elaborada pelo autor

Neste capítulo foi possível observar o funcionamento dos três modos de atuação da arquitetura OrchFlow. Verificou-se ainda que o modo híbrido amplia as possibilidades de uso das redes SDN, permitindo a orquestração de múltiplos subdomínios controlados por diferentes implementações de controladores.

5 Conclusão e Trabalhos Futuros

O OrchFlow possibilita que as aplicações tirem proveito de todas as funcionalidades dos recursos físicos da infraestrutura de uma rede baseada em software de forma automatizada. Buscou-se demonstrar através desta dissertação que a integração entre controladores de diferentes implementações e a sua orquestração pode responder aos problemas de escala que atualmente existem nas redes SDN baseadas no protocolo OpenFlow.

O OrchFlow demonstrou-se capaz de automatizar o processo de programação de múltiplos subdomínios, integrados a um único domínio administrativo, com uma visão global, centralizada, possibilitando o funcionamento nos modos proativo, reativo e híbrido.

O modo proativo apesar de oferecer melhor resposta às aplicações, limita a quantidade máxima de regras ao tamanho total da memória disponível nos *switches* utilizados.

O modo reativo utiliza de forma mais eficiente a quantidade de memória TCAM dos *switches* utilizados, pois as regras podem popular as tabelas de fluxos dos *switches* pelo tempo necessário ao funcionamento da aplicação solicitante, expirando logo após o seu término, permitindo que novas aplicações compartilhem o espaço de memória TCAM dos *switches* do mesmo subdomínio. Porém, o modo reativo gera um atraso no estabelecimento da rota fim-a-fim, podendo prejudicar uma determinada aplicação que necessite de imediata transmissão do fluxo de dados para as redes com muitos subdomínios.

O modo híbrido é capaz de unir a praticidade do modo reativo e a funcionalidade do modo proativo, pois oferece um tempo para o estabelecimento das rotas próximo ao tempo empregado num único subdomínio, além de atuar como o modo reativo, oferecendo melhor uso das memórias TCAM dos *switches* OpenFlow.

Para os trabalhos futuros, pode-se incluir o uso de outros controladores já que nesta dissertação apenas os controladores Floodlight e Ryu foram utilizados. Além disso, novos testes de escalabilidade podem ser realizados a fim de avaliar todo o potencial do OrchFlow em gerenciar domínios administrativos grandes, com milhares de elementos de rede e *hosts*. Possíveis questões de segurança, redundância ou falta de comunicação entre os controladores e o OrchFlow poderão ser tratados. Poderá ser feito também um estudo sobre o uso de novos controladores e a otimização dos códigos já utilizados nos módulos instalados nos controladores atuais e da própria aplicação OrchFlow. Por fim, novas aplicações poderão ser desenvolvidas a partir da base do OrchFlow, envolvendo diferentes protocolos.

Referências

BRADEN, R. et al. Rfc2205-resource reservation protocol (rsvp). *The Internet Engineering Task Force*, 1997. Citado na página 30.

CACIATO, L. E. *Virtualização e Consolidação dos Servidores do Datacenter*. [S.l.]: UNICAMP, Campinas, SP, 2010. Citado na página 22.

CUI, C. et al. Network Functions Virtualisation. *Citeseer*, n. 1, p. 1–16, 2012. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.8583&rep=rep1&type=pdf>. Citado na página 21.

DIJKSTRA, E. A note on two problems in connexion with graphs:(numerische mathematik, _1 (1959), p 269-271). Stichting Mathematisch Centrum, 1959. Citado na página 42.

EIFREM, E. Neo4j-the benefits of graph databases. *no: sql (east)*, 2009. Disponível em: <http://neo4j.com/>. Citado na página 40.

FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures. *Building*, v. 54, p. 162, 2000. Disponível em: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. Citado na página 36.

FRATE, M.; MARCZUK, M. K.; VERDI, F. L. Orchflow - uma ferramenta para orquestração de múltiplos controladores openflow. Anais do XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2016), p. 1221–1228, 2016. Citado na página 24.

GREENBERG, A. et al. A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 35, n. 5, p. 41–54, out. 2005. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1096536.1096541>. Citado na página 21.

HAT, R. OASIS Advanced Message Queuing Protocol OASIS Standard. n. October, p. 0–124, 2012. Citado na página 29.

HUNT, P. et al. Zookeeper: Wait-free coordination for internet-scale systems. In: *USENIX Annual Technical Conference*. [S.l.: s.n.], 2010. v. 8, p. 9. Citado na página 31.

KELLER, E.; REXFORD, J. The "platform as a service"model for networking. In: *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. Berkeley, CA, USA: USENIX Association, 2010. (INM/WREN'10), p. 4–4. Disponível em: <http://dl.acm.org/citation.cfm?id=1863133.1863137>. Citado na página 22.

KOPONEN, T. et al. Onix: A distributed control platform for large-scale production networks. *OSDI, Oct*, p. 1—-6, 2010. Disponível em: <http://dl.acm.org/citation.cfm?id=1924943.1924968>. Citado 2 vezes nas páginas 31 e 32.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. ... *Workshop on Hot Topics in Networks*, p. 1–6, 2010. ISSN 1450304095. Disponível em: <http://dl.acm.org/citation.cfm?id=1868466>. Citado na página 47.

MCKEOWN, N. et al. OpenFlow. *ACM SIGCOMM Computer Communication Review*, v. 38, n. 2, p. 69–74, 2008. ISSN 01464833. Citado na página 22.

MELL, P. M.; GRANCE, T. *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, United States, 2011. Citado na página 21.

OLIVEIRA, R. C. S. d.; VERDI, F. L. Tratando mensagens de arp request em redes sdns. In: SBRC. [S.l.]: X Workshop on Clouds and Applications – WCGA, 2012. Citado na página 42.

PHEMIUS, K.; BOUET, M.; LEGUAY, J. DISCO: Distributed multi-domain SDN controllers. *IEEE/IFIP NOMS 2014 - IEEE/IFIP Network Operations and Management Symposium: Management in a Software Defined World*, 2014. Citado na página 29.

POSTEL, J. et al. Rfc 791: Internet protocol. September, 1981. Citado na página 21.

STRIBLING, J. et al. Flexible , Wide-Area Storage for Distributed Systems with WheelFS. *Proceedings of the 6th {USENIX} {S}ymposium on {N}etworked {S}ystems {D}esign and {I}mplementation ({NSDI} '09)*, p. 43–58, 2009. Citado na página 28.

TIRUMALA, A. et al. Iperf: The tcp/udp bandwidth measurement tool. *htt p://dast. nlanr. net/Projects*, 2005. Citado na página 48.

TOOTOONCHIAN, A.; GANJALI, Y. Hyperflow: a distributed control plane for openflow. *Proceedings of the 2010 internet network ...*, p. 3–3, 2010. Disponível em: <http://dl.acm.org/citation.cfm?id=1863133.1863136$>. Citado 2 vezes nas páginas 27 e 28.

TU, C.-C. et al. Cloud-scale data center network architecture. *Monografia*, Citeseer, 2011. Citado na página 21.

YEGANEH, S. H.; GANJALI, Y. Kandoo: a framework for efficient and scalable offloading of control applications. *Proceeding HotSDN '12 Proceedings of the first workshop on Hot topics in software defined networks*, p. 19–24, 2012. Disponível em: <http://dl.acm.org/citation.cfm?id=2342446>. Citado na página 30.

APÊNDICE A – Manual de Instalação

OrchFlow

Uma Ferramenta para Orquestração de
Múltiplos Controladores OpenFlow

OrchFlow
Manual de instalação

UFSCar - Universidade Federal de São Carlos
SOROCABA - 2016

LERIS - Laboratory of Studies in Networks, Innovation and Software
UFSCar - Sorocaba
http://leris.sor.ufscar.br/

Título:
OrchFlow - Uma Ferramenta para Orquestração de Múltiplos Controladores OpenFlow

Evento:
Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)

Fórum:
Salão de Ferramentas do SBRC - 2016

Grupo de trabalho:
Universidade Federal de São Carlos (UFSCar) - Sorocaba
Instituto Federal de São Paulo (IFSP) - Boituva

Participante(s):
Marcelo Frate
Marcelo K. M. Marczuk

Orientador(s):
Fábio L. Verdi

Cópias: 1

Numero de páginas: 7

Data de Publicação:
04 de Abril de 2016

Resumo:

O principal objetivo das redes definidas por software é a centralização da lógica de controle, porém é possível dividir esta lógica entre dois ou mais controladores com o intuito de garantir a escalabilidade. O protocolo OpenFlow define a comunicação entre switches e controladores, entretanto não prevê a comunicação entre controladores, necessária para qualquer tipo de distribuição no plano de controle. Faz-se necessário, portanto, o desenvolvimento de soluções independentes do protocolo, capazes de distribuir essa lógica dentro de um mesmo domínio administrativo. Neste cenário, o OrchFlow surge como uma ferramenta capaz de orquestrar uma rede definida por software, com dois ou mais controladores OpenFlow, permitindo a gerência e o monitoramento da topologia em tempo real.

Sumário

Capítulo 1

Introdução

O OrchFlow atua como um orquestrador, um agente integrador entre as diversas aplicações disponíveis na rede e os diferentes controladores OpenFlow, sob um mesmo controle administrativo, definido aqui como domínio administrativo. O OrchFlow possibilita a comunicação entre as diferentes aplicações através de uma interface Norte, capaz de receber solicitações e invocar serviços pré-determinados. Essas interfaces fazem parte de um único sistema WEB, utilizando *Representational State Transfer* (REST), um protocolo que torna possível a troca de informações entre aplicativos e serviços web, pela qual são solicitados todos os recursos necessários para o estabelecimento de serviços fim a fim. Ao receber tais solicitações, o OrchFlow as processa e de forma orquestrada atua sobre cada um dos controladores conforme o subdomínio a ser alcançado.

1.1 Requisitos

- Apache Tomcat8;
- Banco de Dados Neo4j V2.3;
- Controlador Floodlight;
- javac 1.8.0_72;
- Mininet V 2.2.1;
- Módulo ARPReply;
- Módulo Reactive;
- OpenFlow V 1.3;
- Oracle VirtualBox
- ovs-vsctl (Open vSwitch) 2.0.2;
- Sistema Operacional Linux;

Capítulo 2

Instalação do OrchFlow

O servidor OrchFlow aqui será instalado numa máquina virtual (VM):
Para baixar e instalar o Oracle VirtualBox acesse:
https://www.virtualbox.org/

Crie uma VM e instale o Ubuntu Server 14.04.4 LTS.

Para baixar e instalar o Ubuntu acesse:
http://www.ubuntu.com/download/server

Crie o usuário "orchflow"e senha "OrchFlow2016"

Configure a placa de rede eth0 com IP: 192.168.56.1/24

Instale o banco de dados Neo4J

Para baixar e instalar o Neo4J acesse:
http://neo4j.com/download/

Instale o Apache Tomcat 8.0.33 Released

Para baixar e instalar o Tomcat8 acesse:
http://tomcat.apache.org/

Após a instalação, o usuário precisará configurar o Neo4j e o servidor de aplicação Apache Tomcat para o correto funcionamento do OrchFlow. Para tal, siga os seguintes passos:

Após a instalação do Neo4j você precisou configurar a senha de acesso ao banco de dados, porém para evitar maiores dificuldades de configurações, vamos desabilitar a necessidade de uso de senhas para acesso ao Neo4j e modificar alguns limites.

Editar o arquivo: /var/lib/neo4j/conf/neo4j-server.properties
e altere a seguinte linha:
`dbms.security.auth_enabled=true`
para
`dbms.security.auth_enabled=false`

Editar o arquivo: /etc/security/limits.conf

e incluir as seguintes linhas:
neo4j soft nofile 40000
neo4j hard nofile 40000

Editar o arquivo: /etc/pam.d/su
e descomentar, retirar o caractere "#"da seguinte linha:

```
#session    required   pam_limits.so
```

Executar o seguinte comando:

```
#ulimit -n 40000
```

O arquivo de instalação do OrchFlow é disponibilizado em formato .war (Web application ARchive) que pode ser facilmente instalado. O procedimento para implantar a aplicação no servidor Apache Tomcat pode ser visto na Figura 2.1:
Acesse a página do servidor Tomcat através de seu navegador.
http://192.168.56.1:8080/manager/html

Manager App > Deploy > WAR File to deploy
Selecione o arquivo orchflow.war e clique em deploy.

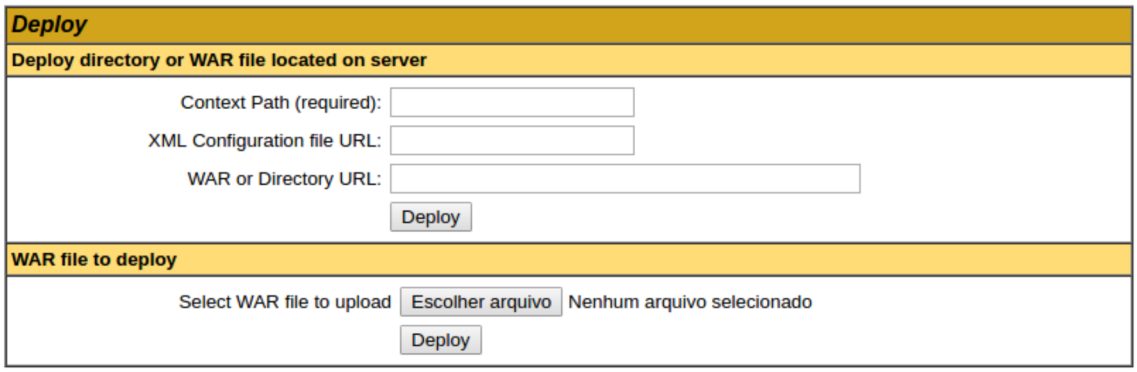


Figura 2.1: Tomcat-deploy.

Após o deploy, será possível administrar a aplicação OrchFlow no servidor Tomcat conforme a Figura 2.2.

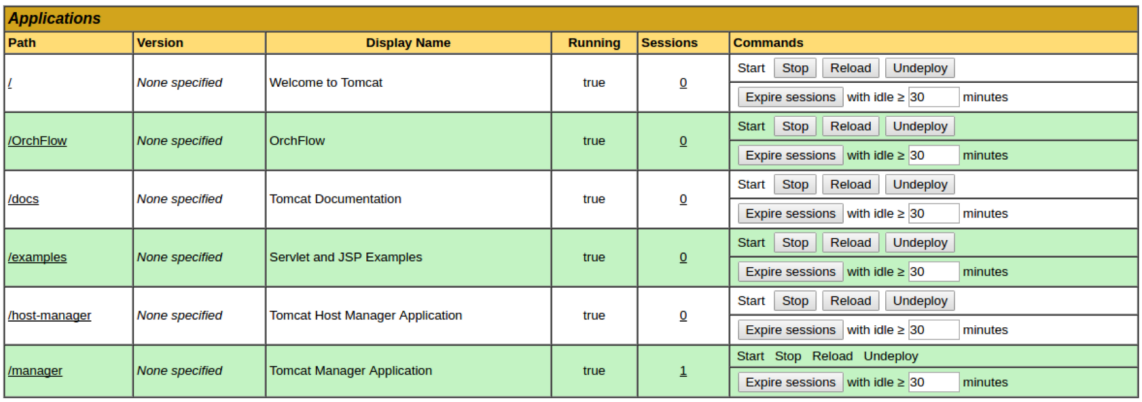


Figura 2.2: Tomcat-applications.

Capítulo 3

Implantando a rede de testes

VM1: Nesta VM, será instalado o Mininet, com o qual o usuário deverá emular a sua rede. Será possível também executar uma rede emulada através de um script preparado com três subdomínios interligados em forma de anel, com 7 switches cada em forma de árvore e 4 hosts, conforme a Figura 3.1, que ilustra a topologia a ser utilizada nos testes do OrchFlow.

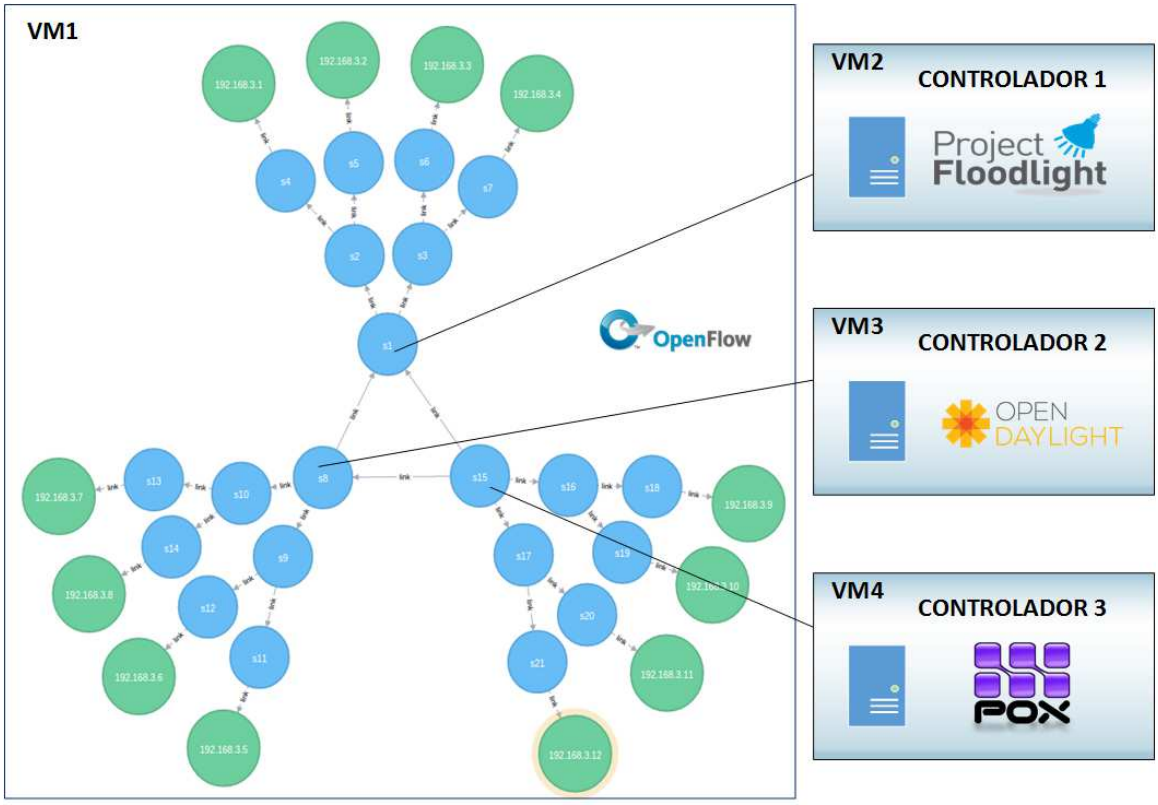


Figura 3.1: Topologia de rede.

Crie uma VM e instale o Ubuntu Server 14.04.4 LTS.

Para baixar e instalar o Ubuntu acesse:
http://www.ubuntu.com/download/server

Crie o usuário "mininet"e senha "mininet"

Configure a placa de rede eth0 com IP: 192.168.56.100/24
Configure a placa de rede eth1 com IP: 10.0.0.100/24

verifique o acesso ao servidor OrchFlow:

```
mininet@mininet:~$ ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1) 56(84) bytes of data.
64 bytes from 192.168.56.1: icmp_seq=1 ttl=64 time=0.314 ms
64 bytes from 192.168.56.1: icmp_seq=2 ttl=64 time=0.248 ms
```

Baixe o arquivo topologia.py e cole na pasta:

```
\home\mininet\mininet\custom
```

Verifique a permissão do arquivo para que possa ser executado:

```
chmod 777 \home\mininet\mininet\custom\topologia.py
```

VM2, VM3 e VM4: Em cada VM, um controlador Floodlight será instalado e configurado para receber as chamadas dos switches das redes instaladas na VM1.

Crie três VMs e instale o Ubuntu Server 14.04.4 LTS.

Para baixar e instalar o Ubuntu acesse:
http://www.ubuntu.com/download/server

Crie o usuário "mininet"e senha "mininet"

Para o controlador 1:
Configure a placa de rede eth0 com IP: 192.168.56.101/24
Configure a placa de rede eth1 com IP: 10.0.0.101/24

Para o controlador 2:
Configure a placa de rede eth0 com IP: 192.168.56.102/24
Configure a placa de rede eth1 com IP: 10.0.0.102/24

Para o controlador 3:
Configure a placa de rede eth0 com IP: 192.168.56.103/24
Configure a placa de rede eth1 com IP: 10.0.0.103/24

verifique o acesso ao servidor OrchFlow em cada controlador:

```
mininet@mininet:~$ ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1) 56(84) bytes of data.
64 bytes from 192.168.56.1: icmp_seq=1 ttl=64 time=0.314 ms
64 bytes from 192.168.56.1: icmp_seq=2 ttl=64 time=0.248 ms
```

verifique o acesso ao mininet em cada controlador:

```
mininet@mininet:~$ ping 192.168.56.100
PING 192.168.56.100 (192.168.56.100) 56(84) bytes of data.
64 bytes from 192.168.56.100: icmp_seq=1 ttl=64 time=0.314 ms
64 bytes from 192.168.56.100: icmp_seq=2 ttl=64 time=0.248 ms
```

A rede 192.168.56.0 será utilizada para o acesso aos serviços de gerenciamento e uso do OrchFlow, por onde toda a configuração dos controladores serão realizadas.

A rede 10.0.0.0 será utilizada pelo protocolo OpenFlow, é a camada de controle da rede.

Para fins de demonstração neste Salão, estamos utilizando apenas controladores Floodlight em cada uma das três VMs.

Para baixar e instalar o Floodlight acesse:
`ttps://floodligt.atlassian.net/wiki/display/floodlightcontroller/Installation+Guide*`

Antes de executar o controlador é preciso instalar e configurar os módulos ARPReply e Reactive.

Para descobrir como criar um módulo para o Floodlight acesse:
`ttps://floodligt.atlassian.net/wiki/display/floodlightcontroller/How+to+Write+a+Module*`

Baixe os arquivos:
ARPReply.zip
Reactive.zip
Descompacte e cole na pasta:
`/home/mininet/floodlight/src/main/java/net/floodlightcontroller`
Você verá as duas pastas arpreply e reactive.

Em seguida, baixe o arquivo:
`net.floodlightcontroller.core.module.IFloodlightModule`
E cole na pasta:
`/home/mininet/floodlight/src/main/resources/META-INF`

Depois, baixe o arquivo:
`floodlightdefault.properties`
E cole na pasta:
`/home/mininet/floodlight/src/main/resources`

E recompile o controlador:
```
root@controlador1:/home/mininet/floodlight# ant;
```
Agora é só rodar o controlador:
```
root@controlador1:/home/orchflow# java -jar /home/orchflow/floodlight/target/floodlight.jar
-cf /home/orchflow/floodlight/src/main/resources/floodlightdefault.properties
```

Apêndice A

Lista de arquivos

O manual do usuário assim como o código fonte dos módulos necessários para Floodlight, os arquivos de configuração e o script da topologia utilizada estão disponíveis em:

`https://github.com/marcelofrate/OrchFlow/`

- ARPReply.zip
- floodlightdefault.properties
- net.floodlightcontroller.core.module.IFloodlightModule
- Reactive.zip
- topologia.py

APÊNDICE B – Manual do Usuário

OrchFlow

Uma Ferramenta para Orquestração de
Múltiplos Controladores OpenFlow

OrchFlow
Manual do usuário

UFSCar - Universidade Federal de São Carlos
SOROCABA - 2016

LERIS - Laboratory of Studies in Networks, Innovation and Software
UFSCar - Sorocaba
http://leris.sor.ufscar.br/

Título:
OrchFlow - Uma Ferramenta para Orquestração de Múltiplos Controladores OpenFlow

Evento:
Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)

Fórum:
Salão de Ferramentas do SBRC - 2016

Grupo de trabalho:
Universidade Federal de São Carlos (UFSCar) - Sorocaba
Instituto Federal de São Paulo (IFSP) - Boituva

Participante(s):
Marcelo Frate
Marcelo K. M. Marczuk

Orientador(s):
Fábio L. Verdi

Cópias: 1

Numero de páginas: 20

Data de Publicação:
04 de Abril de 2016

Resumo:

O principal objetivo das redes definidas por software é a centralização da lógica de controle, porém é possível dividir esta lógica entre dois ou mais controladores com o intuito de garantir a escalabilidade. O protocolo OpenFlow define a comunicação entre switches e controladores, entretanto não prevê a comunicação entre controladores, necessária para qualquer tipo de distribuição no plano de controle. Faz-se necessário, portanto, o desenvolvimento de soluções independentes do protocolo, capazes de distribuir essa lógica dentro de um mesmo domínio administrativo. Neste cenário, o OrchFlow surge como uma ferramenta capaz de orquestrar uma rede definida por software, com dois ou mais controladores OpenFlow, permitindo a gerência e o monitoramento da topologia em tempo real.

Sumário

Capítulo 1

Introdução

OrchFlow, uma ferramenta que tem como objetivo funcionar como um *Middleware*, um software orquestrador para as redes SDN baseadas no protocolo OpenFlow [2], capaz de receber solicitações de serviços através de uma interface Norte *(Northbound)*, processá-las e então mapeá-las através de uma interface Sul *(Southbound)*, de forma a prover o serviço solicitado em uma rede controlada por múltiplos controladores OpenFlow. Quando um determinado serviço é solicitado, o OrchFlow define como deverá ser o tratamento por parte dos diversos controladores até que sejam aplicadas as regras OpenFlow a todos os elementos de rede.

1.1 Arquitetura do OrchFlow

Como se pode ver na Figura 1.1, o OrchFlow atuará como um agente integrador entre as diversas aplicações disponíveis na rede e os diferentes controladores OpenFlow, sob um mesmo controle administrativo, definido aqui como Domínio Administrativo. Para fins de demonstração neste Salão, estamos utilizando apenas controladores Floodlight.

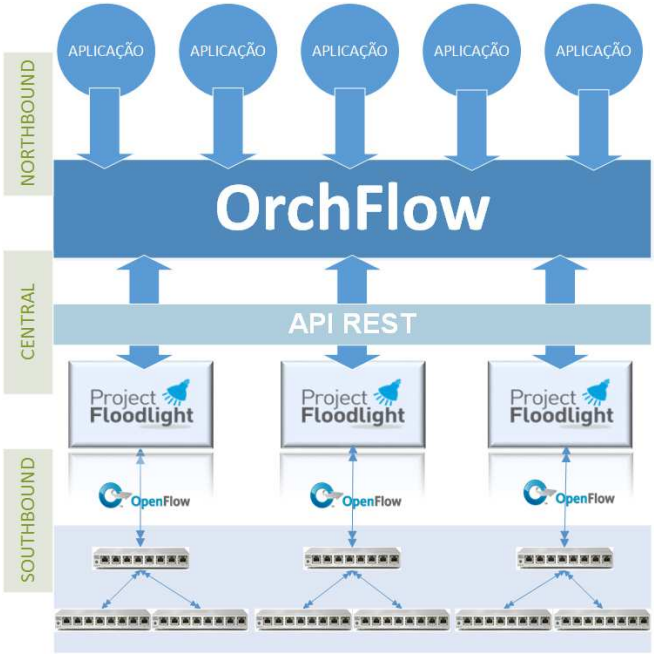


Figura 1.1: Arquitetura utilizada nos testes.

Capítulo 2

Funcionamento do OrchFlow

Como mostrado na Figura 1.1, a primeira camada da arquitetura contém os switches, formando toda a infraestrutura necessária para a comunicação entre os hosts. Para efeito de testes e uso deste manual, optamos por uma rede emulada através do emulador Mininet [1];

2.1 Detalhes da implementação

A arquitetura de rede aqui proposta contém um domínio administrativo composto por três subdomínios, com um controlador OpenFlow cada. Cada controlador é responsável pelo controle e gerenciamento de um único subdomínio e cabe ao OrchFlow a orquestração, gerencia e a visão completa do Domínio Administrativo.

Utiliza-se aqui quatro máquinas virtuais (VM):

VM2, VM3 e VM4: Em cada uma, um controlador Floodlight já está instalado, configurado, funcionando e pronto para receber as chamadas dos switches das redes instaladas na VM1;

VM1: Nesta VM, o usuário deverá emular a sua rede. Para facilitar o acesso e uso, foi preparado um script de uma rede emulada contendo três subdomínios interligados em forma de anel, com 7 switches cada em forma de árvore e 4 hosts.

A Figura 2.1, ilustra a topologia utilizada nos testes e no desenvolvimento do OrchFlow.

Toda a infraestrutura necessária para a implantação e testes do OrchFlow está baseada na plataforma Linux. Assim, o LERIS, disponibiliza um servidor para acesso remoto com todas as máquinas virtuais, com o banco de dados Neo4j e o OrchFlow já instalados e configurados para que todos os testes sejam realizados.

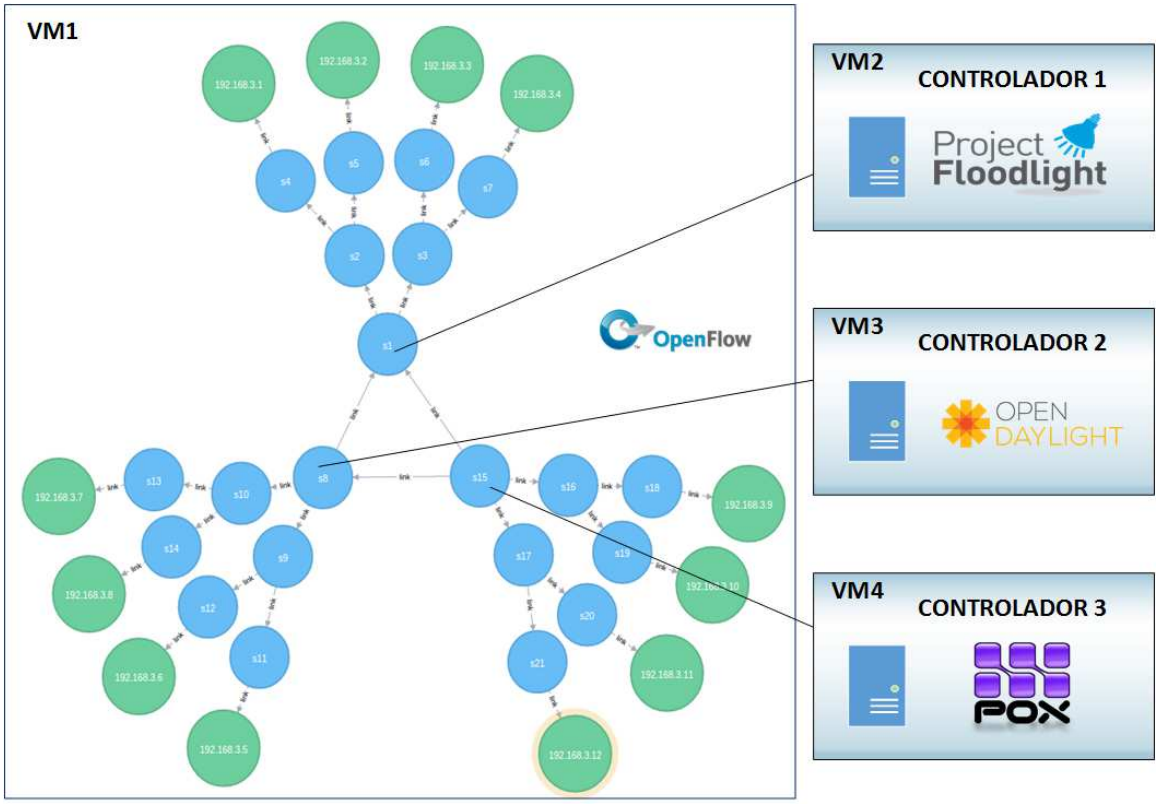


Figura 2.1: Topologia de rede.

Capítulo 3

Requisitos de Software

3.1 Funcionalidades

- Centralização da gestão da rede de computadores;
- Visão global do domínio administrativo;
- Criação de serviços através do modo proativo;
- Criação de serviços através do modo reativo em tempo real;
- Administração dos serviços cadastrados;
- Cadastro e manutenção dos controladores da rede;

3.2 Requisitos

- javac 1.8.0_72;
- Banco de Dados Neo4j V2.3;
- Mininet V 2.2.1;
- OpenFlow V 1.3;
- ovs-vsctl (Open vSwitch) 2.0.2;
- 3 Controladores Floodlight;
- Módulo ARPReply instalado nos 3 controladores;
- Módulo Reactive instalado nos 3 controladores;

Capítulo 4

Inicialização do sistema de testes

Para executar o experimento será necessário o acesso, via SSH, ao servidor do LERIS através do endereço IP: 200.133.238.125, usuário "orchflow"e senha "OrchFlow2016". Como se pode ver na Figura 4.1 este servidor contém 4 máquinas virtuais, criadas especialmente para o Salão de Ferramentas do SBRC 2016. Todas as máquinas estão a disposição do usuário, que poderá acessá-las via SSH com usuário e senha "mininet".

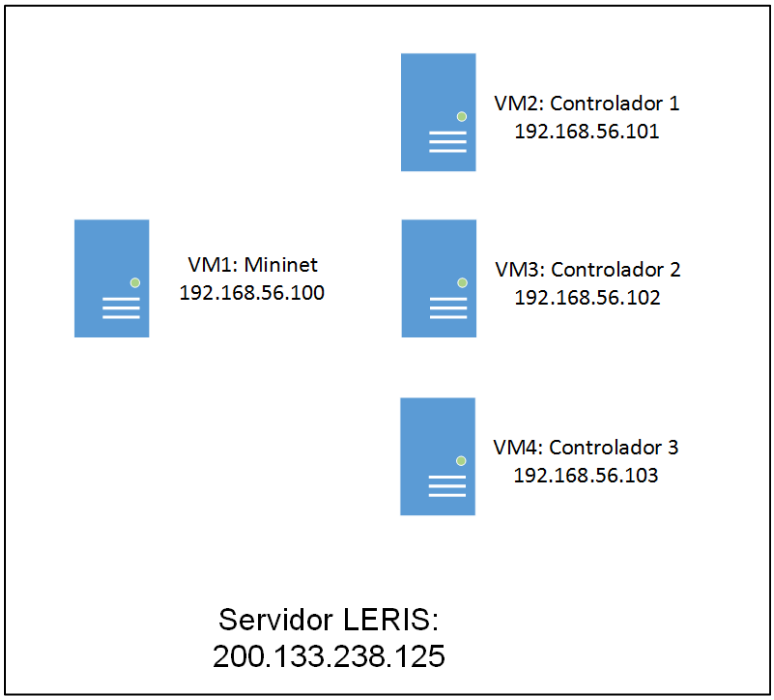


Figura 4.1: Ambiente LERIS.

Para maior facilidade, os três controladores Floodlight já estão sendo executados nas suas respectivas VMs(2, 3 e 4). Ficando portanto ao usuário, a necessidade de carregar apenas a sua topologia na VM1, onde o emulador Mininet já está instalado.

Importante observar que em caso de problemas de acesso ao servidor LERIS, ou erros de funcionamento do OrchFlow, o usuário poderá acessar um servidor de backup, disponibilizado pelo IFSP - Campus Boituva, no endereço IP: **200.133.218.83** com as mesmas configurações de acesso incluindo usuários e senhas.

4.1 Acesso ao Servidor LERIS

Para a realização dos testes recomendamos usar um terminal de console linux, porém, o acesso poderá ser feito através de programas como o PUTTY em outros sistemas operacionais.

Para acessar o servidor LERIS abra um terminal e execute os seguintes comandos:

Acesso ao servidor LERIS

```
ssh -X orchflow@200.133.238.125
orchflow@200.133.238.125's password: OrchFlow2016
orchflow@ServerLeris:~$
```

Recomenda-se o comando ssh -X, pois permite ao usuário orchflow rodar programas com interfaces gráficas, como por exemplo o navegador firefox que será utilizado para acessar o *dashboard* dos controladores. Estando com acesso ao servidor, você poderá acessar a 1ª máquina virtual (VM1), onde está instalado o Mininet e emular a sua própria rede ou executar um script preparado especificamente para este evento.

VM1: Mininet

```
orchflow@ServerLeris:~$ ssh -X mininet@192.168.56.100
mininet@192.168.56.101's password: mininet
mininet@mininet:~$ sudo mn -c
[sudo] password for mininet: mininet
mininet@mininet:~$ sudo ./mininet/custom/topologia.py
```

O comando "mn -c" está sendo utilizado para limpar possíveis configurações deixadas pelo mininet e como dito anteriormente, o script topologia.py foi criado especificamente para este evento. Caso o usuário queira criar a sua própria topologia, este terá a liberdade de fazê-la, porém, deve-se observar que as quatro VMs se comunicam através da rede interna 10.0.0.0.

O script utilizado para criar a topologia segue no Anexo A.

Espere até que seja executado todo o script. Ele criará 12 (doze) hosts, 21 (vinte e um) switches, 3 (três) conexões com os controladores e 30 (trinta) links internos, entre os switches e hosts, além dos 3 (três) links externos que fazem a ligação entre os subdomínios.

Por fim, o script fará com que cada host tente executar um ping simples para um host qualquer, a fim de permitir que os controladores identifiquem cada um dos hosts na rede. Esse procedimento é necessário para demonstrarmos o correto funcionamento do OrchFlow, pois o modo *forward* dos controladores foi desabilitado e é este modo que faz o reconhecimento automático de cada host.

Quando aparecer o prompt do mininet (mininet>) significa que o script executou completamente, que a rede está completa e pronta para o início dos testes.

Capítulo 5

Acessando o sistema OrchFlow

Para acessar o OrchFlow, abra o seu navegador preferido e digite o seguinte endereço:

`http://200.133.238.125:8080/OrchFlow/`

A página inicial do OrchFlow abrirá conforme a Figura 5.1, clique no botão iniciar.

Figura 5.1: Página Inicial.

É importante lembrar que o OrchFlow é uma aplicação WEB, instalada no servidor LERIS utilizando o Apache Tomcat e que poderá ser reiniciada em caso de problemas técnicos.

Na página seguinte, como se pode ver na Figura 5.2, o usuário deverá cadastrar os controladores utilizados na sua rede. Para o nosso experimento, os endereços utilizados são:

Controlador 1: 192.168.56.101 na porta 8085
Controlador 2: 192.168.56.102 na porta 8085
Controlador 3: 192.168.56.103 na porta 8085

Figura 5.2: Configurar.

Note que estes são os endereços das máquinas virtuais criadas específicamente para este evento. Ao terminar o cadastro dos controladores clique em Concluir.

O OrchFlow fará a busca pelos 3 controladores cadastrados e solicitará através da interface REST de cada um dos controladores os dados referentes a topologia de cada subdomínio, cadastrando-os no banco de dados Neo4j, previamente instalado e acessível pelo endereço:

http://200.133.238.125:7474/browser/

Note porém, que não há a necessidade de acessar o banco de dados de forma externa ao OrchFlow, pois como se pode ver nas Figuras 6.1 e 6.2, ele já dispõe de uma interface Neo4j integrada, do lado direito da página, onde é possível ver a topologia completa do domínio administrativo e executar os comandos desse banco de dados, tais como: mudança de cor para os nós e arestas, mudança no tamanho dos nós e identificadores. É possível realizar também comandos de busca de melhor caminho entre dois hosts. Em resumo, todos os comandos do Neo4j estão livres para uso e você pode ver como em:
`http://neo4j.com/`
`http://neo4j.com/docs/stable/`

Do lado esquerdo da interface é possível identificar os dois modos de operação do OrchFlow, proativo e reativo e criar as regras conforme a necessidade da aplicação. Para ambos os modos o procedimento é o mesmo, basta preencher os campos necessários para a definição do serviço fim a fim.

Além disso, é possível observar no menu as opções de gerência dos serviços estabelecidos para os dois modos. Cada um deles afetará de forma diferente os switches, pois as regras no modo proativo são gravadas diretamente nas tabelas de fluxos dos switches, enquanto as regras no modo reativo são tabelas gravadas nos controladores que responderão aos switches sempre que novas conexões chegarem.

Por fim, é possível reconfigurar os controladores caso tenha ocorrido algum erro de digitação na tela anterior.

Capítulo 6

Utilizando o OrchFlow

Para a realização dos testes, um dos serviços de rede mais conhecidos está sendo proposto como forma de validar esta ferramenta: o serviço de roteamento fim a fim.

Pode-se iniciar os testes através do modo **Proativo** onde toda a programação é executada pelo OrchFlow e transferida aos controladores, via interface REST. Neste modo, todas as regras de fluxos são enviadas imediatamente aos switches envolvidos no serviço a ser estabelecido.

Após isso, pode-se realizar os mesmos testes no modo **Reativo** onde as regras são criadas e enviadas para os controladores, porém, diferentemente do modelo Proativo, elas não são implantadas imediatamente nos switches. Tais regras serão implantadas em resposta a um evento *packet-in*. Este evento é enviado através da interface Sul (OpenFlow) para o controlador do subdomínio. O controlador fará então uma busca em sua programação e, caso haja uma correspondência, ele adiciona as regras nas tabelas de fluxos de todos os switches de seu subdomínio, necessários para o estabelecimento do serviço solicitado.

Em ambos os casos, ao clicar no menu Criar rota proativa ou reativa, você terá os campos OpenFlow para determinar as características de cada um dos fluxos, Figuras 6.1 e 6.2, Aqui você deverá escolher o host de origem e o host de destino, criar um nome para a rota e definir os parâmetros seguintes conforme a necessidade.

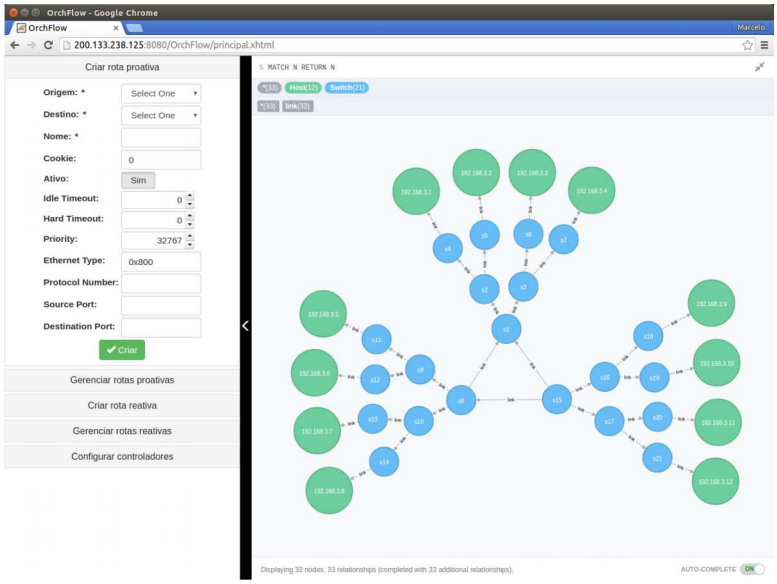


Figura 6.1: Proativo.

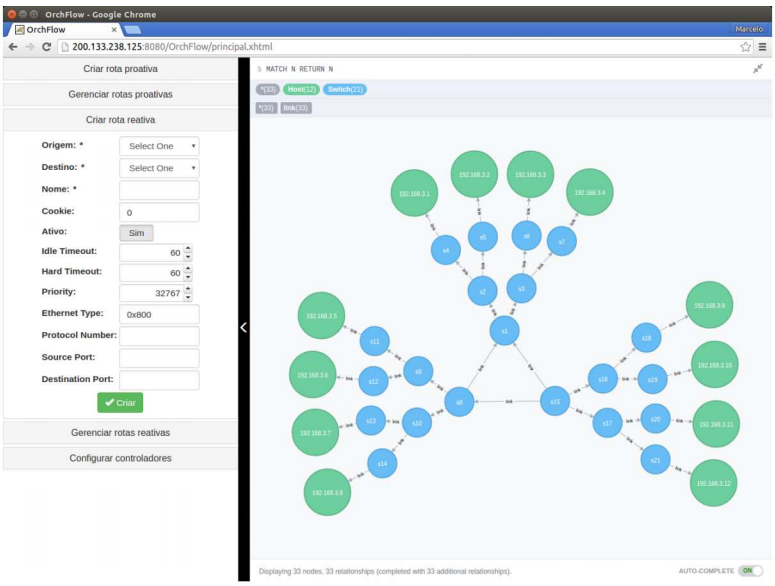


Figura 6.2: Reativo.

Deixamos alguns exemplos de caso de uso, que servirão como modelo para o usuário caso queira criar ou realizar os seus próprios experimentos.

6.1 Ping

Para efeito de teste de conectividade, é possível criar uma rota com regras específicas e demonstrar através do comando ping. É possível testar a conectividade entre dois hosts de subdomínios diferentes como por exemplo entre o host 9 e o host 2, nos dois modos de operação, proativo e reativo.

Primeiro o usuário precisa escolher o modo (aqui escolhemos o modo proativo, Fig. 6.1) e depois configurar o OrchFlow com os seguintes parâmetros:

Origem: 192.168.3.9
Destino: 192.168.3.2
Nome: PING-9-2
Cookie: 0
Ativo: Sim
Idle Timeout: 0
Hard Timeout: 0
Priority: 32767
Ethernet Type: 0x800
Protocol Number: 0x01
Porta Origem:
Porta Destino:

Tal configuração permitirá o tráfego ICMP, (*Protocol Number*: 0x01), entre os dois hosts escolhidos, hosts estes de duas redes diferentes. Assim, o ping funcionará de um subdomínio para outro, independentemente de origem e destino. Para verificar se a programação foi aplicada corretamente o usuário poderá voltar ao terminal onde está sendo executado o mininet e digitar o seguinte comando:

h9 ping h2
É possível executar também:
h2 ping h9
Espera-se obter um resultado parecido com o da figura 6.3

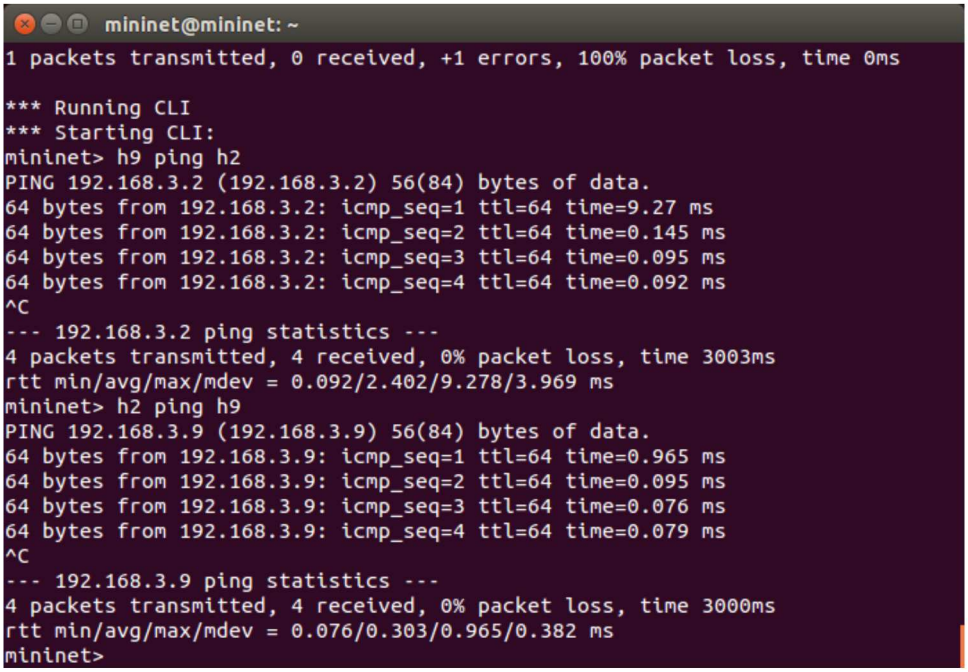


Figura 6.3: Ping entre h9 e h2.

Percebe-se aqui que o OrchFlow configurou através dos controladores todos os seis switches (s5, s2, s1, s15, s16 e s18) que compõem o caminho entre os hosts h9 e h2, criando em suas tabelas de fluxos os caminhos de ida e volta. É possível verificar essa configuração diretamente nos controladores, através de suas interfaces. Para isso, abra um novo terminal, acesse novamente o servidor LERIS e abra o navegador firefox do servidor. Você terá acesso aos endereços dos dashboard dos controladores e poderá ver todas as configurações criadas para cada subdomínio.

Acesso ao servidor LERIS

```
ssh -X orchflow@200.133.238.125
orchflow@200.133.238.125's password: orchflow
orchflow@ServerLeris:~$ firefox &
```

Tal comando fará com que abra o navegador firefox do servidor LERIS conforme a Figura 6.4 com 3 abas já direcionadas para os 3 controladores, onde você terá acesso a cada subdomínio e poderá obter as informações sobre as regras em cada um dos switches desse subdomínio. Pode-se observar também especificamente no switch número 5 que as regras de ida e volta foram criadas para os hosts 9 e 2, bem como em todos os outros switches envolvidos no serviço de estabelecimento da rota fim a fim.

6.2 SSH

Para este serviço, iremos configurar o OrchFlow no modo reativo com os seguintes parâmetros:

Origem: 192.168.3.8

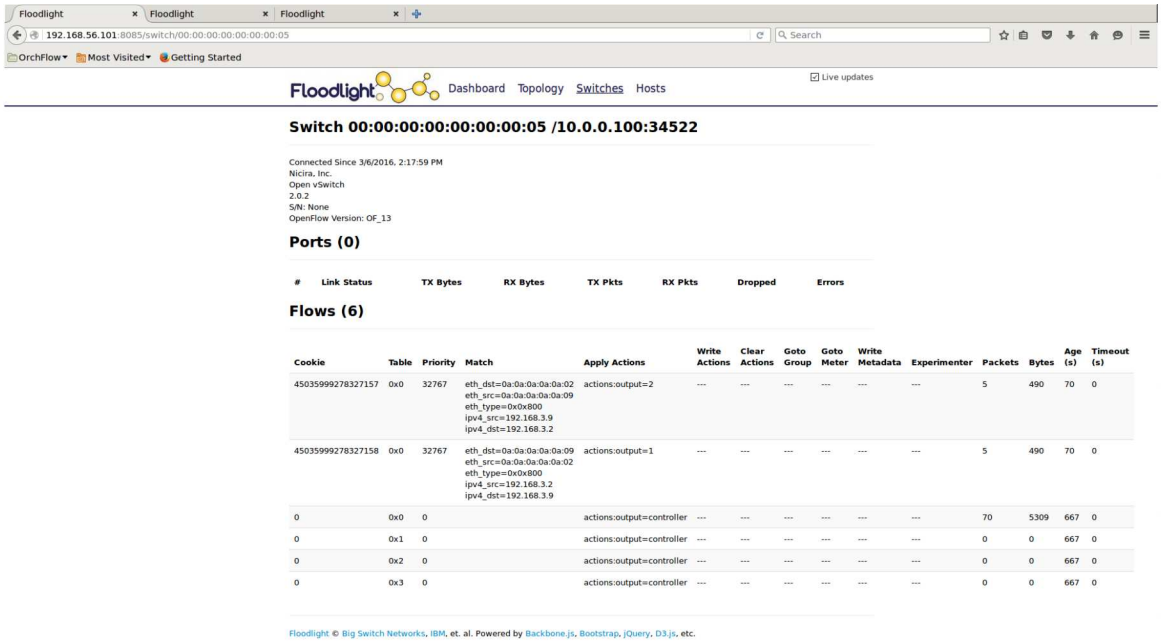


Figura 6.4: Controlador 1 - SW05

Destino: 192.168.3.1
Nome: SSH1
Cookie: 0
Ativo: Sim
Idle Timeout: 60
Hard Timeout: 60
Priority: 32767
Ethernet Type: 0x800
Protocol Number: 0x6
Porta Origem:
Porta Destino: 22

Repare que a porta origem ficará vazia, pois poderá vir de qualquer porta o pedido de conexão com o servidor.

Uma vez que tenha sido executado o comando os controladores receberão as programações devidas, porém, as regras ainda não estarão configuradas nos switches. Pode-se verificar diretamente nos switches (s4, s2, s1, s8, s10, s14).

Volte ao terminal do mininet para acessar os hosts e proceda com as seguintes etapas de configurações e comandos:

Para o host 1 que funcionará como um servidor, foi preciso habilitar o serviço, verificando em qual porta o serviço seria executado e possibilitando que o usuário root obtivesse acesso, para isso, foi editado o arquivo:
`/etc/ssh/sshd_config`

Foi observada a existência das seguintes linhas:

port 22
`#AllowUsers`

PermitRootLogin yes

O arquivo foi salvo e reiniciado o serviço SSH no host, através dos comandos.

```
# /etc/init.d/ssh stop
# /etc/init.d/ssh start
```

Note, que o usuário não precisará mais executar os procedimentos acima, pois neste experimento os hosts já estão preparados.

Entretanto, para o host 8 que funcionará como um cliente, o usuário precisará acessá-lo e prosseguir com o acesso ao host 1, servidor, através dos seguintes comandos:

No terminal do Mininet, utilize o comando:
xterm h8
Abrirá uma nova janela de terminal conforme a Figura 6.5, para o host 8 que terá acesso ao host 1 através do comando:
`#ssh root@192.168.3.1`

A senha de acesso ao host é "mininet".

Um detalhe interessante é que nenhum outro serviço estará habilitado para esta rota, nem mesmo o ping.

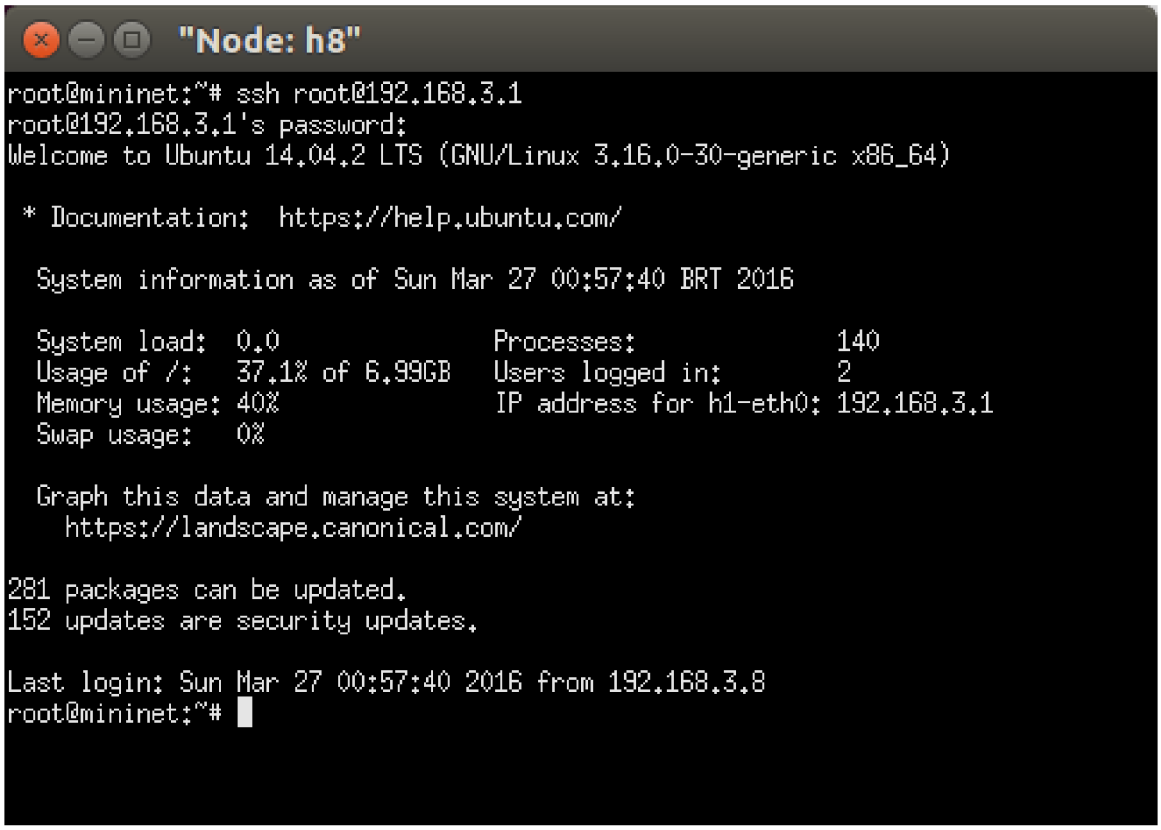


Figura 6.5: Serviço SSH

6.3 Servidor HTTP

Configure o OrchFlow no modo reativo com os seguintes parâmetros:

Origem: 192.168.3.10
Destino: 192.168.3.3
Nome: HTTP3
Cookie: 0
Ativo: Sim
Idle Timeout: 60
Hard Timeout: 60
Priority: 32767
Ethernet Type: 0x800
Protocol Number: 0x6
Porta Origem:
Porta Destino: 80

Repare que a porta origem ficará vazia, pois poderá vir de qualquer porta o pedido de conexão com o servidor.

Uma vez que tenha sido executado o comando os controladores receberão as programações devidas, porém, as regras ainda não estarão configuradas nos switches.

Acesse os hosts proceda com as seguintes configurações e comandos:

No host 3 que funcionará como um servidor, é preciso habilitar o serviço executando o seguinte comando:

```
#python -m SimpleHTTPServer 80 &
```

no host 8, que funcionará aqui como o cliente, basta executar o comando:

```
#wget -O - 192.168.3.3
```

se tudo correr bem, o usuário receberá uma confirmação:

Connecting to 192.168.3.3:80... connected.
HTTP request sent, awaiting response... 200 OK

Um detalhe interessante é que nenhum outro serviço estará habilitado para esta rota, nem mesmo o ping.

Caso queira finalizar o serviço no servidor, utilize o seguinte comando:

```
#kill %python
```

6.4 Servidor FTP

Configure o OrchFlow no modo proativo com os seguintes parâmetros:

Origem: 192.168.3.11
Destino: 192.168.3.4

Nome: FTP4
Cookie: 0
Ativo: Sim
Idle Timeout: 60
Hard Timeout: 60
Priority: 32767
Ethernet Type: 0x800
Protocol Number: 0x6
Porta Origem:
Porta Destino: 21

Repare que a porta origem ficará vazia, pois poderá vir de qualquer porta o pedido de conexão com o servidor.

Uma vez que tenha sido executado o comando os controladores receberão as programações devidas, encaminhando as regras diretamente aos switches. Percebe-se aqui que o OrchFlow configurou através dos controladores todos os seis switches (s20, s17, s15, s1, s3 e s7) que compõem o caminho entre os hosts h11 e h4, criando em suas tabelas de fluxos os caminhos de ida e volta. É possível verificar essa configuração diretamente nos controladores, através de suas interfaces.

Acesse os hosts proceda com as seguintes configurações e comandos:

No host 4 que funcionará como um servidor, é preciso habilitar o serviço.

```
# inetd &
```

no host 11, que funcionará como um cliente, basta executar o comando:

```
#ftp 192.168.3.4
```

o usuário e a senha de acesso ao host é mininet.

Um detalhe interessante é que nenhum outro serviço estará habilitado para esta rota, nem mesmo o ping.

Bibliografia

[1] Bob Lantz, Brandon Heller e N McKeown. "A network in a laptop: rapid prototyping for software-defined networks". Em: ... *Workshop on Hot Topics in Networks* (2010), pp. 1–6. ISSN: 1450304095. DOI: 10.1145/1868447.1868466. URL: http://dl.acm.org/citation.cfm?id=1868466.

[2] Nick McKeown et al. "OpenFlow". Em: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74. ISSN: 01464833. DOI: 10.1145/1355734.1355746.

Apêndice A

Topologia da Rede

```
#!/usr/bin/python

import re
import sys

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSKernelSwitch, UserSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info, error
from mininet.link import Link, TCLink, Intf
from mininet.util import quietRun

def topology():
"Create a network."
net = Mininet( controller=RemoteController, link=TCLink, switch=OVSKernelSwitch )

print "*** Creating nodes"
h1 = net.addHost( 'h1', mac='0a:0a:0a:0a:0a:01', ip='192.168.3.1/24' )
h2 = net.addHost( 'h2', mac='0a:0a:0a:0a:0a:02', ip='192.168.3.2/24' )
h3 = net.addHost( 'h3', mac='0a:0a:0a:0a:0a:03', ip='192.168.3.3/24' )
h4 = net.addHost( 'h4', mac='0a:0a:0a:0a:0a:04', ip='192.168.3.4/24' )

h5 = net.addHost( 'h5', mac='0a:0a:0a:0a:0a:05', ip='192.168.3.5/24' )
h6 = net.addHost( 'h6', mac='0a:0a:0a:0a:0a:06', ip='192.168.3.6/24' )
h7 = net.addHost( 'h7', mac='0a:0a:0a:0a:0a:07', ip='192.168.3.7/24' )
h8 = net.addHost( 'h8', mac='0a:0a:0a:0a:0a:08', ip='192.168.3.8/24' )

h9 = net.addHost( 'h9', mac='0a:0a:0a:0a:0a:09', ip='192.168.3.9/24' )
h10 = net.addHost( 'h10', mac='0a:0a:0a:0a:0a:0a', ip='192.168.3.10/24' )
h11 = net.addHost( 'h11', mac='0a:0a:0a:0a:0a:0b', ip='192.168.3.11/24' )
h12 = net.addHost( 'h12', mac='0a:0a:0a:0a:0a:0c', ip='192.168.3.12/24' )

s1 = net.addSwitch( 's1', protocols='OpenFlow13', listenPort=6671, mac='00:00:00:00:00:01' )
s2 = net.addSwitch( 's2', protocols='OpenFlow13', listenPort=6672, mac='00:00:00:00:00:02' )
s3 = net.addSwitch( 's3', protocols='OpenFlow13', listenPort=6673, mac='00:00:00:00:00:03' )
```

```
s4 = net.addSwitch( 's4', protocols='OpenFlow13', listenPort=6674, mac='00:00:00:00:00:04' )
s5 = net.addSwitch( 's5', protocols='OpenFlow13', listenPort=6675, mac='00:00:00:00:00:05' )
s6 = net.addSwitch( 's6', protocols='OpenFlow13', listenPort=6676, mac='00:00:00:00:00:06' )
s7 = net.addSwitch( 's7', protocols='OpenFlow13', listenPort=6677, mac='00:00:00:00:00:07' )

s8 = net.addSwitch( 's8', protocols='OpenFlow13', listenPort=6678, mac='00:00:00:00:00:08' )
s9 = net.addSwitch( 's9', protocols='OpenFlow13', listenPort=6679, mac='00:00:00:00:00:09' )
s10 = net.addSwitch( 's10', protocols='OpenFlow13', listenPort=6680, mac='00:00:00:00:00:0a' )
s11 = net.addSwitch( 's11', protocols='OpenFlow13', listenPort=6681, mac='00:00:00:00:00:0b' )
s12 = net.addSwitch( 's12', protocols='OpenFlow13', listenPort=6682, mac='00:00:00:00:00:0c' )
s13 = net.addSwitch( 's13', protocols='OpenFlow13', listenPort=6683, mac='00:00:00:00:00:0d' )
s14 = net.addSwitch( 's14', protocols='OpenFlow13', listenPort=6684, mac='00:00:00:00:00:0e' )

s15 = net.addSwitch( 's15', protocols='OpenFlow13', listenPort=6685, mac='00:00:00:00:00:0f' )
s16 = net.addSwitch( 's16', protocols='OpenFlow13', listenPort=6686, mac='00:00:00:00:00:10' )
s17 = net.addSwitch( 's17', protocols='OpenFlow13', listenPort=6687, mac='00:00:00:00:00:11' )
s18 = net.addSwitch( 's18', protocols='OpenFlow13', listenPort=6688, mac='00:00:00:00:00:12' )
s19 = net.addSwitch( 's19', protocols='OpenFlow13', listenPort=6689, mac='00:00:00:00:00:13' )
s20 = net.addSwitch( 's20', protocols='OpenFlow13', listenPort=6690, mac='00:00:00:00:00:14' )
s21 = net.addSwitch( 's21', protocols='OpenFlow13', listenPort=6691, mac='00:00:00:00:00:15' )

c1 = net.addController( 'c1', controller=RemoteController, ip='10.0.0.101', port=6653 )
c2 = net.addController( 'c2', controller=RemoteController, ip='10.0.0.102', port=6653 )
c3 = net.addController( 'c3', controller=RemoteController, ip='10.0.0.103', port=6653 )

print "*** Creating links"
net.addLink(s1, s2, 1, 1)
net.addLink(s1, s3, 2, 1)
net.addLink(s2, s4, 2, 1)
net.addLink(s2, s5, 3, 1)
net.addLink(s3, s6, 2, 1)
net.addLink(s3, s7, 3, 1)

net.addLink(s8, s9, 1, 1)
net.addLink(s8, s10, 2, 1)
net.addLink(s9, s11, 2, 1)
net.addLink(s9, s12, 3, 1)
net.addLink(s10, s13, 2, 1)
net.addLink(s10, s14, 3, 1)

net.addLink(s15, s16, 1, 1)
net.addLink(s15, s17, 2, 1)
net.addLink(s16, s18, 2, 1)
net.addLink(s16, s19, 3, 1)
net.addLink(s17, s20, 2, 1)
net.addLink(s17, s21, 3, 1)

net.addLink(h1, s4, 0, 2)
```

```
net.addLink(h2, s5, 0, 2)
net.addLink(h3, s6, 0, 2)
net.addLink(h4, s7, 0, 2)
net.addLink(h5, s11, 0, 2)
net.addLink(h6, s12, 0, 2)
net.addLink(h7, s13, 0, 2)
net.addLink(h8, s14, 0, 2)
net.addLink(h9, s18, 0, 2)
net.addLink(h10, s19, 0, 2)
net.addLink(h11, s20, 0, 2)
net.addLink(h12, s21, 0, 2)

print "*** Starting network"
net.build()
c1.start()
c2.start()
c3.start()

s1.start( [c1] )
s2.start( [c1] )
s3.start( [c1] )
s4.start( [c1] )
s5.start( [c1] )
s6.start( [c1] )
s7.start( [c1] )

s8.start( [c2] )
s9.start( [c2] )
s10.start( [c2] )
s11.start( [c2] )
s12.start( [c2] )
s13.start( [c2] )
s14.start( [c2] )

s15.start( [c3] )
s16.start( [c3] )
s17.start( [c3] )
s18.start( [c3] )
s19.start( [c3] )
s20.start( [c3] )
s21.start( [c3] )

s1.cmd('ovs-vsctl add-port s1 s1-ext1 – set interface s1-ext1 type=patch options:peer=s8-ext1')
s1.cmd('ovs-vsctl add-port s1 s1-ext3 – set interface s1-ext3 type=patch options:peer=s15-ext3')
s8.cmd('ovs-vsctl add-port s8 s8-ext1 – set interface s8-ext1 type=patch options:peer=s1-ext1')
s8.cmd('ovs-vsctl add-port s8 s8-ext2 – set interface s8-ext2 type=patch options:peer=s15-ext2')
s15.cmd('ovs-vsctl add-port s15 s15-ext2 – set interface s15-ext2 type=patch options:peer=s8-ext2')
```

s15.cmd('ovs-vsctl add-port s15 s15-ext3 – set interface s15-ext3 type=patch options:peer=s1-ext3')

s1.cmdPrint('ovs-vsctl show')
h1.cmdPrint('ping 192.168.3.12 -c 1')
h2.cmdPrint('ping 192.168.3.12 -c 1')
h3.cmdPrint('ping 192.168.3.12 -c 1')
h4.cmdPrint('ping 192.168.3.12 -c 1')
h5.cmdPrint('ping 192.168.3.12 -c 1')
h6.cmdPrint('ping 192.168.3.12 -c 1')
h7.cmdPrint('ping 192.168.3.12 -c 1')
h8.cmdPrint('ping 192.168.3.12 -c 1')
h9.cmdPrint('ping 192.168.3.12 -c 1')
h10.cmdPrint('ping 192.168.3.12 -c 1')
h11.cmdPrint('ping 192.168.3.12 -c 1')
h12.cmdPrint('ping 192.168.3.1 -c 1')

print "*** Running CLI"
CLI(net)

print "*** Stopping network"
net.stop()

```
if __name__ == '__main__':
```
setLogLevel('info')
topology()