Aurelio Grott, Gabriel Dominico, Victor Lucas de M. Mafra

# *Análise e solução de vulnerabilidades em ambiente LAMP baseada em experimentação com Kali Linux*

Joinville

2016

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA**
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**Aurelio Grott, Gabriel Dominico, Victor Lucas de M. Mafra**

# ANÁLISE E SOLUÇÃO DE VULNERABILIDADES EM AMBIENTE LAMP BASEADA EM EXPERIMENTAÇÃO COM KALI LINUX

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

**Charles Christian Miers**
**Orientador**

Joinville, Junho de 2016

# ANÁLISE E SOLUÇÃO DE VULNERABILIDADES EM AMBIENTE LAMP BASEADA EM EXPERIMENTAÇÃO COM KALI LINUX

Aurelio Grott, Gabriel Dominico, Victor Lucas de M. Mafra

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação Integral do CCT/UDESC.

Banca Examinadora

---

Charles Christian Miers - Doutor (orientador)

---

Charles Christian Miers - Doutor

---

Charles Christian Miers - Doutor

# Agradecimentos

*"We must know - we will know!"*

*- David Hilbert*

# Resumo

O Software Defined Network (SDN) é uma tecnologia recente que permite ao administrador de redes um maior controle sobre uma rede. Tal controle é obtido através da separação entre o *Control Plane* e *Data Plane*, o que caracteriza uma SDN. Neste trabalho são conceituados diversos pontos chaves relativos ao assunto, tais como *OpenFlow* e os planos do SDN. Em seguida é descrita a ferramenta de simulação de redes *Mininet* e no fim do trabalho é descrito dois *benchmarks* com o objetivo de coletar dados para uma análise de desempenho. Tendo em vista que os *hardwares* que suportam SDN são relativamente recentes, o seu custo é muitas vezes proibitivo para um pequeno grupo de pesquisa ou um pesquisador independente, de tal forma que o uso de simuladores se torna indispensável para o desenvolvimento cientifico e tecnológico na área. Este trabalho tem como objetivo realizar um comparativo entre a transferência de dados de tamanho médio dentro de um ambiente simulado no *Mininet* e um ambiente utilizando o modelo tradicional de rede (*Data Plane* e *Control Plane* acoplados).

Palavras-chaves: SDN, Openflow, Mininet, software livre, transferência

# Abstract

SDN is a new technology which gives the network administrator greater power over his network. Such control is given through the separation between the control plane and the data plane, which characterizes an SDN. In this paper it is conceptualized several key points relative to SDN, such as *OpenFlow* and the SDN planes. In the following section it is described the networking simulation tool *Mininet* folowwing by the description of two benchmarks that will be used with the objective of data collection for later analisys. Since the hardwares that supports SDN are relatively new, their costs is very often prohibitively high for a small research group or an independent researcher, so that the use of simulators becomes indispensable for the technological and scientific development in the field. This paper has as its major objective to do an comparative between the transferering of medium sized data in a Mininet simulated environment and a traditional networking model (i.e. coupled Data Plane and Control plane).

Keywords: SDN, Openflow, Mininet, open source, trasnference

# Lista de Figuras


2.1 Diagrama da arquitetura de uma SDN. . . . . 13

2.2 Exemplo flwo-table. . . . . 17

2.3 Exemplo flwo-table. . . . . 17

2.4 Topologia da rede do ambiente de testes . . . . . 27

2.5 Taxa de transferência de arquivo de 256 Megabytes no Mininet . . . . . 30

2.6 Taxa de transferência de arquivo de 512 Megabytes no Mininet . . . . . 30

2.7 Taxa de transferência de arquivo de 768 Megabytes no Mininet . . . . . 30

2.8 Taxa de transferência de arquivo de 1024 Megabytes no Mininet . . . . . 30

2.9 Taxa de transferência de arquivo de 256 Megabytes no OpenStack . . . . . 31

2.10 Taxa de transferência de arquivo de 512 Megabytes no OpenStack . . . . . 31

2.11 Taxa de transferência de arquivo de 768 Megabytes no OpenStack . . . . . 31

2.12 Taxa de transferência de arquivo de 1024 Megabytes no OpenStack . . . . 31

2.13 Taxa de transferência de arquivo de 256 Megabytes no ambiente físico . . . 32

2.14 Taxa de transferência de arquivo de 512 Megabytes no ambiente físico . . . 32

2.15 Taxa de transferência de arquivo de 768 Megabytes no ambiente físico . . . 32

2.16 Taxa de transferência de arquivo de 1024 Megabytes no ambiente físico . . 32

2.17 Taxa, média e desvio padrão de transferência no ambiente físico . . . . . 34

2.18 Taxa, média e desvio padrão de transferência no ambiente Mininet . . . . . 34

2.19 Comportamento anormal no Mininet . . . . . 35

2.20 Consumo de RAM pelo Mininet . . . . . 35

2.21 Tempos de boot do Mininet . . . . . 35

# Lista de Tabelas

2.1 Tamanho dos arquivos de teste . . . . . 26

2.2 Configuração da máquina virtual para o teste *Mininet* . . . . . 28

2.3 Configuração da máquina virtual para o teste de rede tradicional . . . . . 28

2.4 Configuração da máquina física para o teste de rede tradicional . . . . . 29

2.5 Taxa de transferência média . . . . . 33

2.6 Desvio padrão da taxa de transferência . . . . . 33

# Lista de Siglas e Abreviaturas

**API** Aplication Programming Interface

**CLI** Command Line Interface

**CPU** Central Processing Unit

**DPCF** Data Plane Control Function

**IETF** Internet Engineering Task Force

**I/O** Input/Output

**GNU** Gnu Not Unix

**LSD** Link State Database

**NFS** Network File System

**ONF** Open Networking Fundation

**OSPF** Open Shortest Path First

**PRNG** Pseudo-Random Number Generator

**QOS** Quality of Service

**RAM** Random Access Memory

**SDN** Software Defined Network

**SNMP** Simple Network Management Protocol

**UDESC** Universidade do Estado de Santa Catarina

# Sumário

**Lista de Figuras** **5**

**Lista de Tabelas** **6**

**Lista de Siglas e Abreviaturas** **7**

**1 Introdução** **10**

**2 Conceitos** **11**

2.0.1 Arquitetura . . . . . 12

2.0.2 Data plane e Control plane . . . . . 14

2.0.3 Management plane . . . . . 16

2.0.4 Data Plane . . . . . 16

2.0.5 Openflow . . . . . 16

2.0.6 Funcionamento do OpenFlow . . . . . 18

2.0.7 Principais benefícios do OpenFlow . . . . . 18

2.0.8 Exemplo de uso do OpenFlow . . . . . 18

2.1 Mininet . . . . . 19

2.1.1 Arquitetura . . . . . 19

2.1.2 Funcionamento . . . . . 20

2.1.3 API Python . . . . . 21

2.1.4 Desvantagens . . . . . 22

2.2 Análise . . . . . 24

2.2.1 Benchmarks da aplicação . . . . . 25

2.2.2 Benchmarks das redes simuladas . . . . . 25

2.2.3 Ambiente de testes . . . . . 27
2.2.4 Experimentos . . . . . 29
2.2.5 Análise dos resultados . . . . . 30
2.3 Conclusão . . . . . 36

**Referências Bibliográficas** **37**

# 1 Introdução

Atualmente, com a grande explosão de crescimento da Internet e do *Cloud Computing*, as redes de computadores se tornaram cada vez mais complexas, demandando cada vez mais esforços para gerenciá-las. Com o objetivo de melhorar e faciliar o gerenciamento de grandes redes, um novo paradigma de rede foi introduzido, o *Software Defined Network* (SDN). O SDN caracteriza-se pela separação nítida do *Data Plane* do *Control Plane*. Tal separação permite que o administrador tenha o controle de sua rede através de uma camada de abstração, permitindo distribuir e controlar os recursos de sua rede de um ponto logicamente centralizado sem ter que acessar individualmente cada dispositivo de rede. O uso do SDN deu ao administrador de redes a capacidade de alterar qualquer porção da rede assim que necessário, seja a alteração uma priorização, redirecionamento ou bloqueamento de pacotes em específico. Isto tornou-se essencial nos ambientes modernos de *Cloud Computing* onde volumes inéditos de informação estão disponíveis.

Para que o uso do SDN cresça, além da divulgação e acessibilidade, é necessario promover o desenvolvimento cientifico e tecnológico. Atualmente existe uma organização dedicada a promover o a promoção e adoção do SDN, a *Open Networking Foundation* (ONF). A ONF foi responsável por apresentar e desenvolver o *OpenFlow*, protocolo que fornece uma abstração dos recursos de rede. O *OpenFlow* é tido como a primeira padronização aberta de SDN e é considerada um elemento central no desenvolvimento.

O SDN é uma tecnologia relativamente nova no mercado, e isto se traduz em preços elevados, muitas vezes proibitivos para universidades e outras instituições que promovem o desenvolvimento científico e tecnológico. Como alternativa ao uso de *Hardware* compativel com SDN, existe a possibilidade da utilização de simuladores e emuladores tais como o *Mininet*. No entanto é preciso conhecer suas capacidades e validar seus resultados. Este trabalho tem como objetivo comparar o desempenho fornececido pelo *Mininet* com uma situação real de transferência de arquivos de médio porte.

# 2 Conceitos

A informação viaja pelo mundo na forma de pacotes digitais, passando por vários roteadores e switches, quase sempre em redes *IP*. Apesar de serem as mais adotadas, as redes *IP* são *complexas e difíceis de administrar*. Para conseguir configurar as redes com as políticas de alto nível desejadas, cada dispositivo de rede precisa ser configurado utilizando comandos de baixo nível e algumas vezes específicos do fornecedor. Além da complexidade da configuração, as redes precisam suportar e se adaptar às mudanças de carga. Por isso, é muito difícil cumprir com as políticas de rede desejadas. Para dificultar ainda mais, as redes atuais integram o *Control Plane*(que decide como lidar com o tráfego de rede) e o *Data Plane*(que encaminha o tráfego de acordo com as decisões tomadas pelo *Control Plane*).

As redes estabelecem uma conexão baseada em um conjunto de critérios para identificar o menor, mais rápido ou mais seguro caminho (*Control Plane*) e movem os dados através dessas conexões (*Data Plane*). Quando o arcabouço de rede foi originalmente estabelecido, houve um considerável esforço em construir camadas no *Data Plane*. Essas camadas o tornam simples, rápido e eficiente para efetuar mudanças dentro de uma camada, sem afetar as demais no mesmo plano. Infelizmente, não houve uma sistematização de camadas similar no *Control Plane*. Ao invés disso, existem numerosos protocolos que decidem como estabelecer conexões, que foram criados ao passar dos anos, adicionando cada vez mais complexidade. Ao separar o *Control Plane* do *Data Plane*, o funcionamento do *Control Plane* pode ser modificado sem afetar o que já funciona corretamente no *Data Plane*. A premissa fundamental do *Software Defined Networking* é separar a tomada de decisão no *Control Plane* da execução dessas decisões no *Data Plane* (ONF, 2012). A *Open Network Foundation* (*ONF*) define *Software Defined Networking* (*SDN*) como a separação física do plano de controle de rede (*Control Plane*) do plano de encaminhamento (*Data Plane*), no qual um plano de controle controla vários dispositivos. O plano de controle exerce controle direto sobre os elementos da rede (por exemplo, roteadores, switches) através de uma *Application Programming Interface* bem definida (*API*).

Embora o termo *SDN* tenha sido utilizado inicialmente para descrever o projeto *OpenFlow* de Stanford, a sua definição expandiu para incluir uma gama muito mais

ampla de tecnologias. Assim, para entendermos melhor a história do *SDN*, apenas destacamos a evolução nas relações das ideias que evidenciam as características do *SDN*, sem tentar encontrar uma relação direta entre os projetos. Segundo (FEAMSTER; ZEGURA, 2011), a história pode ser dividida em três estágios, cada um com sua contribuição: (1) Redes Ativas (meados de 1990 ao começo de 2000), que introduziram funções programáveis na rede, na qual os roteadores intermediários podem processar os pacotes que passam através deles, e até mesmo modificar tais pacotes, ao invés de somente entregar pacotes de um ponto a outro como nas redes passivas. (2) Separação do *Control Plane* e *Data Plane* (perto de 2001 até 2007), que desenvolveu interfaces abertas entre o *Control Plane* e o *Data Plane* como por exemplo o *ForCES* (*Forwarding and Control Element Separation*), uma interface padronizada pela *Internet Engineering Task Force* (*IETF*). (3) *OpenFlow API* e sistemas operacionais de rede (por exemplo, *OpenSwitch*).

### 2.0.1 Arquitetura

Com base nos dados apresentados pela *The Open Networking Fundation* (ONF) (ONF, 2014b), que é a fundação responsável na padronização da SDN, podemos afirmar que a arquitetura de uma SDN consiste na dissociação do plano de controle e de dados, a centralização da inteligência e a monitoração do estado atual da rede, deixando abstrata a infraestrutura para a aplicação. A Figura 2.1 ilustra a arquitetura funcional de uma rede SDN.

Como pode ser observado na Figura 2.1 a arquitetura ONF/SDN é definida através de 3 camadas, que são:

- **A Camada de Aplicação** consiste nas aplicações criadas pelo programador, esta camada consome os serviços de comunicação SDN. A comunicação entre a camada de aplicação e a camada de controle é feita através da *Northbound API*.
- **A Camada de Controle** fornece a funcionalidade de controle consolidada, que supervisiona o comportamento de fluxo da rede através de uma interface aberta.
- **A Camada de Infraestrutura** consiste nos elementos da rede (do inglês *Network Elements* ou NE) e dispositivos que oferecem a comutação e o encaminhamento de pacotes. A comunicação entre a camada de infraestrutura e a camada de controle é feita através da *Northbound API*.

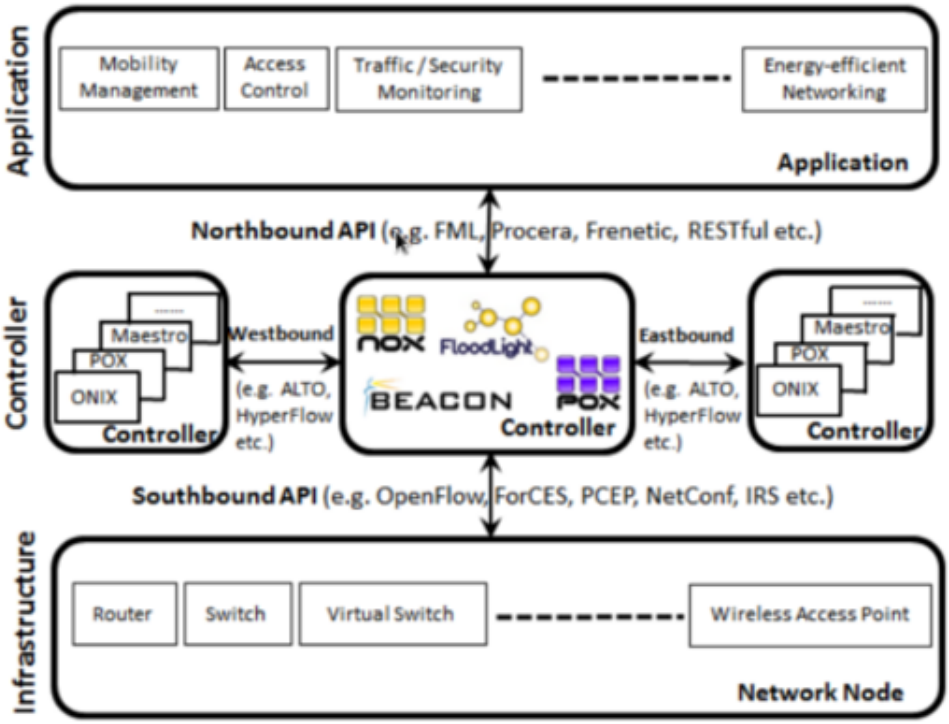


Figura 2.1: Diagrama da arquitetura de uma SDN.

Seguindo esse modelo de camadas estabelecido pela ONF (ONF, 2014b), podemos caracterizar a arquitetura de SDN dentro dos seguinte atributos:

- **Inteligência logicamente centralizada.** Na arquitetura SDN, o controle da rede é distribuído a partir do encaminhamento usando a interface já padronizada: Openflow. Através da inteligência de rede centralizada, a tomada de decisão é facilmente baseada em uma visão global da rede, ao contrário das redes de hoje, que são construídas com base em sistemas autônomos, onde os nós não têm conhecimento do estado global da rede.

- **Programabilidade.** Redes SDN são controladas através de funcionalidades de softwares, que podem ser fornecidos pelos vendedores, ou pelos próprios operadores de rede. Essa programabilidade permite que o paradigma de gestão possa ser substituído por processos automatizados, influenciados pela rápida adoção da nuvem.

- **Abstração** Em uma rede SDN, as aplicações que consomem serviços SDN são abstraídas para as tecnologias de rede subjacentes. Dispositivos de rede também são abstraídos para a camada de controle, afim de garantir a portabilidade e futuro para investimentos em serviços de rede, onde o software de rede reside na camada de controle.

### 2.0.2 Data plane e Control plane

Um *Switch* ou *router* é constituído de duas partes principais: O *Control plane* e o *Data plane* (ONF, 2013). O *Control Plane* é o componente responsável por controlar a maneira pela qual o *Switch* se comunica com os seus vizinhos, e também por decidir o próximo *hop* de cada pacote recebido e prover informações para *hosts* adjacentes. O *Data Plane* recebe os pacotes e os reenvia conforme sua *Forwarding table*, que foi construída com informações recebidas do *Control Plane*.

#### Open Shortest Path First

O *Open Shortest Path First* (OSPF) é um protocolo utilizado pelo *Control Plane*, no modelo tradicional de rede para rotear pacotes. As aplicações do *OSPF* se extendem para a construção automatizada da tabela de roteamento de uma *SDN*, caso desejável. O OSPF utiliza uma tabela de vizinhança (Neighbor table), um *Link state database* (LSD) e uma tabela de roteamento de IPs.

Cada *switch* da rede mantém um banco de dados idêntico descrevendo a topologia de rede, onde cada entrada do banco de dados corresponde ao estado de um outro *switch* na rede. Este banco de dados é denominado LSD. Cada *switch* comunica seu estado aos demais através de *flooding*. Para cada *switch* adjacente na rede existe uma entrada em uma tabela de vizinhança, a qual é utilizada para determinar os dispositivos alcançáveis e associar cada *host* adjacente com um uma porta física do dispositivo.

Com a informação contida na LSD constrói-se um grafo direcionado que corresponde a topologia da rede e a direção de fluxo. Considerando que cada nó da rede possui o mesmo LSD, tem-se que cada nó possuirá a mesma representação de grafo. Tendo o grafo a disposição, o *switch* calcula uma arvore com os menores custos para cada nó da rede, tendo a si mesma como a raiz da arvore.

#### Control Plane

O *Control plane* utiliza um algoritmo de roteamento para estabelecer a maneira na qual é feito o *fowarding* dos pacotes. No caso de uma SDN, conforme já citado anteriormente, tem-se acesso ao *Control plane* de forma a redefinir o trafego da rede. Os pacotes que chegam no *switch* podem ser dados para serem redirecionados ou informações relevantes

ao *control plane* oriundas do *management plane* ou de outro *switch*. Para o caso de ser um pacote com destino outro que o *switch*, o pacote é redirecionado de acordo com a tabela de roteamento do *data plane*, do contrario o pacote é interpretado pelo *control plane*.

De maneira geral o *control plane* se comunica com outros *switchs* na rede e usa as informações coletadas juntamente com as definições do *management plane* para criar uma tabela de roteamento. As aplicações do *control plane* utilizam significativamente menos recursos que o *data plane* e não necessitam de um tempo de resposta tão baixo.

A especificação feita pela *Open Networking Fundation* (ONF) considera o *Control Plane* como sendo uma caixa preta, definido pelo seu comportamento externamente observável (ONF, 2014a). O fabricante do dispositivo de rede esta livre para decidir como os papéis do *Data Plane* serão implementados. Quatro componentes básicos compõem o *data plane*. O *data plane control function* (DPCF), *coordinator*, *virtualizer*, e o agente.

### Data Plane control function

O DPCF é responsável por comandar os recursos disponíveis para um dispositivo de rede e os usa conforme determinado pelo *coordinator* ou o *virtualizer*.

### Coordinator

O *coordinator* é o elemento da SDN que efetua os comandos emitidos pelo gerente da rede. O alcance do *coordinator* é universal, já que a gerência é realizada em cima do *data plane*, *control plane* e do *application plane*.

### Virtualizer

Um controlador SDN oferece serviços às suas aplicações por via de um modelo de informação de instância que é derivado dos recursos da rede, políticas de gerenciamento e funções de suporte locais ou externas. A entidade funcional que suporta a instancia de modelo de informação é denomina *virtualizer* (ONF, 2014a).

**Agent**

O *agent* do *control plane* é a entidade abstrata que representa as capacidades do servidor e os recursos do cliente. Considerando a estrutura recursiva do SDN, o controlador em um nível $N$ representa os recursos e ações disponíveis em um nível $N + 1$ (ONF, 2014a).

### 2.0.3 Management plane

O *management plane* consiste em uma interface direta ou indireta com o administrador da rede pela qual é feito o *network shaping*. SSH, Telnet e SNMP são exemplos de protocolos utilizados para a comunicação com o *management plane*.

### 2.0.4 Data Plane

O *data plane* é responsável por inspecionar os pacotes e envia-los para a interface de rede necessária de mode que o pacote em questão chegue ao seu destino (KREUTZ et al., 2014). Dentro do *data plane* existe uma tabela de roteamento que é feita com informações providas pelo *control plane* e o *management plane*.

A parte mais crítica do *switch* é o *data plane*, já que ali que esta a responsabilidade de inspecionar e chavear de forma adequada cada pacote. Todo o *Quality Of Service* (*QOS*), filtragem, *queuing*, encapsulamento, *parsing* de cabeçalhos e *policying* é executado no *data plane*.

### 2.0.5 Openflow

O OpenFlow, proposto pela Universidade de Stanford é um protocolo implementado tanto nos dispositivos de infraestrutura de rede, quanto no controlador SDN. O principal conceito por trás do OpenFlow é a criação de flow-tables (tabelas de fluxo) nos roteadores, onde um fluxo é constituído pela combinação de campos do cabeçalho do pacote a ser processado pelo dispositivo. Um modelo de flow-table, que ainda está sujeito a melhorias, é mostrado na Figura 2.2.

Um switch Openflow é constituído de ao menos três partes: (1) Uma *Flow Table*, que com uma ação associada a cada fluxo, guia o switch como processar o fluxo, (2) Um canal seguro, que conecta o switch ao controlador, aceitando comandos e pacotes

| MAC src | MAC dst | IP Src | IP Dst | TCP dport | ... | Action | Count |
|---|---|---|---|---|---|---|---|
| * | 10:20:. | * | * | * | * | port 1 | 250 |
| * | * | * | 5.6.7.8 | * | * | port 2 | 300 |
| * | * | * | * | 25 | * | drop | 892 |
| * | * | * | 192.* | * | * | local | 120 |
| * | * | * | * | * | * | controller | 11 |

Figura 2.2: Exemplo flwo-table.

que serão enviados entre o controlador e o próprio switch (3) O *Protocolo OpenFlow*, que providencia uma maneira padronizada para que o controlador se comunique com o switch (MCKEOWN et al., 2008).

O canal seguro garante a segurança durante a troca de informações entre o switch e o controlador. É recomendado como interface de acesso o protocolo SSL (Secure Socket Layer).

Como protocolo, o OpenFlow é aberto (open soruce) e fica responsável por estabelecer a comunicação entre os dispositivos de rede e controladores.

O Controlador é um software que pode rodar em um servidor qualquer e fica responsável por adicionar e remover entradas nas flow-tables, uma SDN pode conter inúmeros controladores para obter mais performance e robustez. Na Figura 2.3, podemos observar uma rede com OpenFlow.

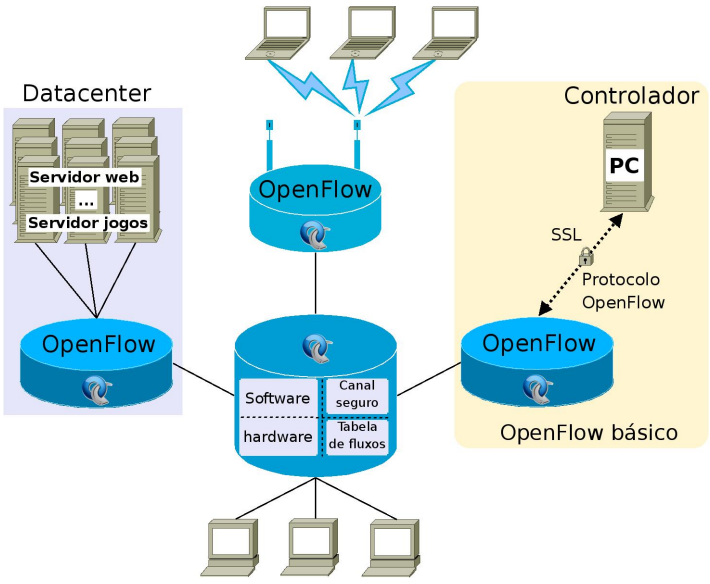


Figura 2.3: Exemplo flwo-table.

### 2.0.6 Funcionamento do OpenFlow

O pacote de dados, que chega a um equipamento OpenFlow, tem seus cabeçalhos comparados com as regras de entrada das flow-tables, se existir alguma regra em que o pacote se encaixe, então os contadores (utilizados para a remoção de fluxos inativos e obtenção de estatísticas) são atualizados e através da regra associada, é possível executar uma ação previamente estabelecida para o pacote, que define a forma como o pacote deve ser processado e para onde ele deve ser encaminhado. Caso o pacote não possua uma regra associada na tabela de fluxos, o seu tratamento fica sob responsabilidade do controlador SDN, esse que por sua vez pode instalar uma nova regra no *switch* para que os próximos pacotes desse fluxo não precisem todos ser encaminhados ao controlador, ou ainda instalar um regra que obrigue o encaminhamento de todos os pacotes do mesmo fluxo ao controlador. Geralmente isso acontece com os pacotes de controle (ICMP, DNS, DHCP) ou protocolos de roteamento (OSPF, BGP).

### 2.0.7 Principais benefícios do OpenFlow

1. Os mais modernos routers e switches possuem flow-tables que implementam firewalls, NAT, QoS, entre outros serviços. Com a existência dessas flow-tables nos dispositivos de rede, não se faz necessário a troca de hardware para implementar um switch OpenFlow, uma simples troca de firmware é capaz de habilitar as funções necessárias.

2. Com a separação do plano de controle e do plano de dados, é possível a utilização de controladores remotos para a rede, permitindo a possibilidade de terciarização do controle sobre o plano de dados.

3. O OpenFlow é totalmente compatível ao estado atual da internet.

### 2.0.8 Exemplo de uso do OpenFlow

Com a possibilidade de programar redes através do protocolo OpenFlow, diversos experimentos podem ser executados, dentre eles podemos citar (MCKEOWN et al., 2008):

- **Novos protocolos de roteamento:** No OpenFlow, é possível implementar protocolos de roteamento para executarem no controlador da rede. Assim, quando

um pacote chega ao *switch* ele é encaminhado ao controlador, que por sua vez determinará o fluxo do pacote, instalando regras nas tabelas de fluxo dos switches pertencentes a rota do pacote, afim de que não seja necessário o encaminhamento de todos os pacotes do fluxo ao controlador.

- **Mobilidade:** Um cliente WiFi da rede OpenFlow pode estar se movimentando na área de abrangência da infraestrutura WiFi. Assim, podem ser implementados mecanismos de *handoff* no controlador, afim de que o controlador fique responsável por determinar a localidade do cliente dentro da infraestrutura e altere dinamicamente as tabelas de fluxo para criação de uma melhor rota e mudança de um *acess point* para outro através do *handoff*.

## 2.1 Mininet

O Mininet é um emulador de SDN criado com o objetivo de impulsionar o desenvolvimento do OpenFlow (ANTONENKO; SMELYANSKIY, 2013) e capaz de criar redes e dispositivos virtuais em uma única máquina. A abordagem utilizada pelo Mininet para a emulação de prototipagem de redes, utilizando os recursos de virtualização providos pelo sistema operacional permite que sejam executadas redes com centenas de dispositivos de rede (LANTZ; HELLER; MCKEOWN, 2010). Devido ao fato de que o Mininet utiliza de redes virtuais providas pelo Sistema Operacional, é possível implementar novas funcionalidades para a rede ou novos protocolos, testa-las em largas topologias e implantar o mesmo código em uma rede de produção.

O Mininet é distribuído sob uma licença *opensource* própia baseada na licença BSD (BSD..., 2015), a licença é denomina *Mininet License* (MININET..., 2015a). O uso da licença BSD permite que pesquisadores e demais interessados copiem, modifiquem e distribuam o código, promovendo assim o desenvolvimento do Mininet.

### 2.1.1 Arquitetura

O Mininet utiliza recursos do sistema operacional para criar redes e dispositivos virtuais. Nos sistemas GNU/Linux, utiliza-se *network namespaces* para a criação das redes virtuais. Isto provê flexibilidade e segurança, já que as redes criadas pelo Mininet estão logicamente

isoladas da rede do sistema hospedeiro e cada *network namespace* possui sua própia cópia da *stack* TCP/IP (IP-NETNS..., 2015). Isso ainda provê a possibilidade extra de integração do Mininet com redes reais (OLIVEIRA et al., 2013).

A utilização do *Linux Container Architecture* permite executar os dispositivos virtuais dentro de uma *jail*, provendo assim um isolamento do sistema hospedeiro. Recursos tais como CPU, Networking e RAM são providos através de um sistema de virtualização destes, proporcionando um controle fino da utilização de recursos e permitindo a imposição de limite no uso de tais recursos.

### 2.1.2 Funcionamento

Uma das maneiras de se interagir com o Mininet é através de um *command line interface* (CLI) onde se é capaz de criar, interagir e customizar em tempo real protótipos de SDN. O Mininet deve ser executado em modo super usuário devido a sua interação com os recursos fornecidos pelo sistema.

Por padrão, quando iniciado o Mininet cria uma rede de topologia denominada *Minimal*. Uma rede *Mininal* é constituido de duas Máquinas Virtuais conectadas através de um único *switch Openflow* que é gerenciado por um controlador *Openflow*. Topologias mais complexas estão também definidas nativamente no Mininet. As topologias são:

**Single** A topologia *Single* consiste de um *switch Openflow* central e $k$ *hosts*. É criado automaticamente um *link* entre o *switch* e cada host. O $k$-ésimo *host* esta conectado na $k$-ésima porta do *switch*;

**Reversed** A topologia *Reversed* consiste de um *switch Openflow* central e $k$ *hosts* semelhante a topologias *single*. A diferença esta na ordem na qual os *hosts* estão ligados ao *switch*. O $k$-ésimo *host* esta ligado na primeira porta e o primeiro host é ligado a $k$-ésima porta.

**Linear** Na topologia *Linear* tem-se $k$ *hosts* e $k$ *switchs*. O $k$-ésimo *host* é ligado ao $k$-ésimo *switch* e o *switch* na $k$-ésima posição esta conectado aos *switchs* na posição $(k-1)$ e $(k+1)$.

**Tree** A topologia *tree* contém $k$ *hosts* cada um ligado a um *switch* em um nível lógico $n$. Estes *switchs* são ligados dois a dois a um *switch* de nível logico $n-1$, e assim

sucessivamente até que se tenha apenas um *switch* no nível 1. Uma topologia *tree* com $n$ níveis tem $k = 2^n$ *hosts*.

O *Shell* interativo do *Mininet* também permite a execução de comandos direcionados a um único *host*. Isso permite o *fine tuning* de um *host* de maneira semelhante a real, onde o administrador possui acesso direto (física ou remotamente), de tal forma que é possível executar rotinas de sistema ou até mesmo programas de terceiros. Um cenário plausível é a execução de um servidor *Web* em um dos *hosts* e simular o acesso simultâneo através de outros *hosts* conectados á rede. Existem múltiplos comandos que são *built-in* no interpretador de comandos, estes estão documentados na *API mininet* (MININET..., 2015b). Outros comandos disponíveis no sistema hospedeiro onde o *Mininet* está sendo executado e que estejam listados no *PATH* do sistema podem ser executados adicionando o prefixo *sh*.

Para cada *link* da rede tem-se ainda a possibilidade de ajustar diversos parâmetros para serem utilizados durante a simulação do enlace. Estão disponíveis o controle de latência de cada *link* assim como do *bandwidth*. O controle global pode ser realizado através de *flags* durante a chamada do *Mininet* ou interativamente dentro do *Shell*. O controle individual está disponível apenas através da *API Python* do *Mininet*.

### 2.1.3 API Python

A implementação do *Mininet* se baseia principalmente na linguagme *Python* na versão 2.7, contendo apenas alguns pequenos módulos escrito em linguagem C. Ao passo que é possível interagir diretamente com o *Mininet* através de seu *Shell* interativo, a sua utilidade está limitada às chamadas de algumas rotinas de efeito global, como por exemplo alteração da latência, e a execução em alguma máquina virtualizada dos comandos disponíveis no *Shell* do sistema operacional hospedeiro.

Visando à construção de ambiente robusto e flexível de simulação, o *Mininet* provê uma *API* em *Python* para permitir a execução e automatização de tarefas. O *Mininet* juntamente com a sua *API Python* suporta a criação de topologias paramétricas com uma facilidade relativamente grande. A construção de topologias paramétricas permite que possam ser criadas topologias semelhantes em estrutura porém variando em tamanho. O uso é feito criando-se um arquivo com extensão *py* onde deverá ser sobrecarregado o

método *__main__* e invocar os métodos *addSwitch*, *addHost* e *addLink* para adicionar *Switchs*, *Hosts* e *Links* à topologia. A maneira como os elementos da rede estão conectados é determinado pelos argumentos passados à chamada de função. As topologias criadas com a *API Python* podem variar desde topologias simples como estrela e anel até estruturas complexas baseadas em grafos.

A *API Python* ainda fornece outras funcionalidades além da criação de topologias paramétricas, o *tunning* de parâmetros de simulação do enlace. É possível simular latência, perda de pacotes e alterar a largura de banda dos *links* de rede. Os principais parâmetros sujeitos a ajuste são latência, que é dado em unidades de tempo, largura de banda em Mbits, tamanho máximo da fila em numero de pacotes e porcentagem de perda de pacotes.

Finalmente, para se obter o máximo de automatização, é possível realizar chamadas a comandos do *Shell* do sistema hospedeiro. Como cada *host* dentro do mininet é um *Shell*, a execução de comandos se dá passando uma *string* de maneira igual ao que seria executado em uma situação real. A chamada é realizada utilizando-se do comando *cmd*. É interessante ressaltar que o *Mininet* não garante que os processos criados pela chamada do comando *cmd* serão finalizados ao se encerrar o *Mininet*. Cabe ao usuário gerenciar a abertura e fechamento de processos. O *Mininet* provê recursos para esperar algum processo encerrar através da chamada de função *wait*, e possui também a capacidade de esperar por um tempo arbitrário por um processo através deta mesma função. Outro recurso provido pela *API* é o redirecionamento de informação pelo *stdin*, *stdout* e *stderr*, além do uso de *pipes*. Em conjunto com funcionalidades internas da *API*, tais como *multipoll*, permitem que sejam monitorados o *stdout* e *stderr* de vários processos em execução e reagir à determinadas saídas.

### 2.1.4 Desvantagens

Os recursos disponíveis virtualmente pelo *Mininet* são todos executados via *software*, o que impõe um limite artificial ao máximo de *hosts* e *links* disponíveis na topologia a ser simulada. Tendo em vista que todos os controladores virtuais competem pelo mesmo tempo de *CPU* do sistema hospedeiro, em simulações relativamente grandes e com alto fluxo de pacotes, pode apresenhtar um desempenho inferior ao esperado quando comparado a uma situação real. O mesmo pode vir a acontecer caso exista uma ou mais aplicações com alto

consumo de tempo de *CPU*, o que poderia vir a prejudicar o desempenho do enlace simulado. O *Mininet* provê os recursos para compartilhar o ambiente de testes com terceiros, de tal forma que o teste ou simulação realizados podem ser reproduzidos por outras pessoas. No entanto, por motivos supracitados, resultados podem varias caso exista uma diferença de poder computacional entre o sistema hospedeiro de ambos os usuários do *Mininet*.

Outro ponto interessante de se destacar, é que apesar de cada máquina virtualizada pelo *Mininet* rodar isoladamente em um *Linux Container*, todas estão sobre o mesmo *PID* do sistema. Isto não é benéfico pois diversas aplicações não permitem a execução de instâncias simultâneas em um mesmo local. No caso do *Mininet*, aplicações executando em diferentes *hosts* teriam o *PID* do processo do simulador, consequentemente alguns serviçõe pode se recusar a executar por identificar erroneamente a existencia de instância em execução. O *Mininet* provê recursos atravéz de sua *API* para evitar esse problema.

Existe ainda uma limitação extra imposta pela utilização extensiva de recursos do kernel do Linux que é a incapacidade de executar software limitados a sistemas Windows ou MacOS. Diversos recursos de *containers*, *quotas* e *network namespaces* utilizados pelo *Mininet* são exclusivos para o kernel *Linux* ou incompatível com a implementação de outros sistemas operacionais. Existe no entanto a possibilidade de se integrar o *Mininet* com a rede física do sistema hospedeiro ou com a de uma máquina virtual. Isto no entanto adiciona limites extras na reprodutibilidade dos experimentos, já que o o protótipo deixara de ser autocontido no *Mininet*.

O modelo de sistema de arquivos utilizado pelo *Mininet* pode vir a ser fonte de problemas, devido a sua estrutura compartilhada. O compartilhamento do sistema de arquivos fornece diversas vantagens, como minimizar o uso de espaço em disco, evitar a copia de arquivos que são idênticos em uma ou mais máquinas virtualizadas e diminuir o trafego de rede ao compartilhar aquivos. No entanto, diversos serviços utilizam por padrão a configuração no diretório de sistema */etc*, o que impossibilita a execução de diversos serviços *out-of-the-box*. Para evitar esse problema, é necessário criar diretórios unicamente para um *host* em específico através da *API* provida pelo *Mininet*. Outro potencial problema, é ao se executar um mesmo *script* em vários *hosts* simultaneamente. Se o *script* criar ou escrever em um mesmo arquivo, haverá colisão de arquivos, já que todos os *hosts* estarão acessando o mesmo arquivo. Isso deve ser tratado criando-se um

nome de arquivo único a cada execução ou através da utilização de pastas privadas para cada *host*.

#### Alternativas

Considerando as limitações acima ditadas, em alguns casos é desejável que se tenha alternativas ao *Mininet* para comparação e validação do resultado ou para substituição total. Abaixo sera discutido brevemente duas alternativas ao *Mininet*.

**Maxinet** Baseado no *Mininet*, possui o objetivo de superar as limitações em relação ao número de dispositivos de rede que podem ser simulados. Segundo o autor (WETTE et al., 2014), o *Maxinet* tem como propósito simular grandes redes de *data-centers* e superar o desempenho do *Mininet* através da utilização de vários computadores trabalhando de forma distribuída. O *Maxinet* contém *built-in* um gerador de trafego de rede, baseado em recentes análises de casos reais.

**Mininet-Hifi** O *Mininet-Hifi* (HANDIGOL et al., 2012) é um projeto com a finalidade de melhorar a reprodutibilidade de testes realizados no *Mininet*. Implementado de forma semelhante, utilizando *containers*, o *Mininet-Hifi* utiliza de recursos adicionais para melhorar a reprodutibilidade.

## 2.2 Análise

Nesta seção será descrito os testes utilizados como critérios de análise do *Mininet* seguido da análise em sí. A primeira subseção consiste da descrição do *Benchmark* em relação ao consumo de recursos da máquina onde o *Mininet* está em execução. Em seguida ocorre a descrição dos critérios de avaliação visando à análise do desempenho do *Mininet* na transmissão de dados de tamanho médio. Depois ocorre a descrição do ambiente de testes onde o *Mininet* será executado assim como o ambiente físico de testes que será utilizado como objeto de comparação ao *Mininet*. Finalmente é discutido e apresentado os testes em uma subseção própia e é realizada a análise dos resultados em uma subseção final.

### 2.2.1 Benchmarks da aplicação

Um dos primeiros passos deste trabalho consistiu de se medir o tempo necessário para a inicialização do *Mininet*. Apesar de que os tempos de *boot* não possuem um impacto direto no desempenho do *Mininet*, o tempo necessário para a inicialização do *Mininet* pode ser um fator importante na execução de múltiplos casos de teste ou em grandes cenários.

O primeiro critério de avaliação foi o tempo necessário para criar a topologia especificada. Foram realizados uma série de testes de maneira automatizada utilizando às topologias paramétricas do *Mininet*. As topologias utilizadas foram *Single*,*Reversed*, *Linear* e *Tree*, que correspondem às topologias disponíveis por padrão.

Mediu-se o tempo necessário para a criação de cada topologia utilizando o comando time (BASH..., 2015) disponível nos sistemas *GNU/Linux* e na família *BSD*. Os testes da topologia *Single*, *Linear* e *Reversed* foram testados utilizando-se os mesmos parâmetros de testes. O teste consistiu na criação das topologias mencionadas utilizando-se um valor $n$ como parâmetro para a topologia paramétrica. O valor de $n$ iniciou em 1, seguido de 32 com um incremento de 32 até atingir $n = 1024$. Para a topologia *Tree* os valores de $n$ foram organizados de maneira diferenciada devido à natureza da topologia. Considerando um único *host* por nó folha, uma árvore de $k$ níveis possui $2^k$ *hosts*. Foram criados topologias em árvore variando de 1 até 10 níveis, de tal forma que quando $n = 10$ tem-se 1024 *hosts*, fazendo assim com que ao final do teste de cada topologia tenha-se o mesmo número de *hosts* criados.

Ao mesmo tempo que se mediu o tempo necessário para a criação das topologias no *Mininet*, mediu-se o consumo de *RAM* do sistema onde o *Mininet* estava em execução. A medição foi executada utilizando o comando de sistema *free* (FREE(1)..., 2015), que é capaz de mostrar o consumo total de *RAM* do sistema. Antes, e imediatamente após a execução do *Mininet*.

### 2.2.2 Benchmarks das redes simuladas

O teste de performance consistiu na transferência de arquivos entre diferentes *hosts* virtuais dentro do *Mininet*.

Os arquivos foram criados com conteudo aleatório utilizando-se o *Pseudo-*

*Random Number Generator* (PRNG) do sistema operacional onde os testes foram realizados. Utilizou-se 9 arquivos de conteudo aleatório com tamanho variando de 8 Megabytes até 1024 Megabytes. Os tamanhos de cada arquivos estão disponíveis na tabela 2.1.

| Nome do arquivo | Tamanho |
|---|---|
| file_8 | 8 Megabytes |
| file_16 | 16 Megabytes |
| file_32 | 32 Megabytes |
| file_64 | 64 Megabytes |
| file_128 | 128 Megabytes |
| file_256 | 256 Megabytes |
| file_512 | 512 Megabytes |
| file_768 | 768 Megabytes |
| file_1024 | 1024 Megabytes |

Tabela 2.1: Tamanho dos arquivos de teste

Com o objetivo de eliminar interferência de fatores externos à transferência dos arquivos de testes, todos os arquivos antes de serem copiados foram indicados ao sistema operacional como candidato a ser armazenado no cache de I/O. Para aumentar as chances de que o arquivo seja se fato carregado no cache limpou-se toda a memória reservada ao cache. A limpeza foi realizada setando a variável drop_caches do *kernel* Linux acessada em */proc/sys/vm/drop_caches* para o valor 3. Conforme a documentação do kernel Linux (LINUX..., 2015a) marcando a variável drop_caches para o valor 3 faz com que os *slab objects* (LINUX..., 2015b) e *pagecache* sejam limpos.

Para transferir os arquivos de teste entre os *hosts* utilizou-se o *rsync*, *software* capaz de copiar e sincronizar arquivos locais e remotos. Para transferir os arquivos para um local remoto foi configurado um túnel *ssh* entre os locais de origem e destino. Esta mesma configuração foi utlizada nos 3 ambientes de testes.

Como critério de avaliação mediu-se o tempo necessário para a transferência dos arquivos de testes entre dois *hosts* da rede. A medida de tempo foi realizada pelo própio *software* responsável pela tranferencia dos arquivos. A transferência foi efetuata utilizando *rsync* (RSYNC(1)..., 2015). Outro critério de avaliação utilizado foi o monitoramento da taxa de transferência. O monitoramento foi efetuado capturando a saida padrão do *rsync* e salvando-o em um arquivo temporário para análise posterior.

### 2.2.3 Ambiente de testes

Os testes consistiram da execução de ambos os *benchmarks* especificados. Cada *benchmark* foi realizado em três ambientes distintos com o objetivo de coletar dados para realizar um comparativo entre o ambiente real e o ambiente simulado.

#### Topologia da rede de testes

A topologia da rede onde os testes foram realizadas trata-se da topologia *Single* disponível por padrão no *Mininet*. Conforme já citado, a topologia consiste de $k$ roteadores conectados em linha, onde o $k-ésimo$ roteador está conectado com o $(k+1)-ésimo$ roteador e cada roteador está conectado com $n$ *hosts*. No caso do ambiente utilizado para os testes $k = 3$ e $n = 1$. A figura 2.2.3 contém uma representação visual da topologia escolhida.

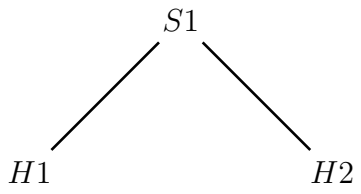


Figura 2.4: Topologia da rede do ambiente de testes

#### Ambiente de simulação do *Mininet*

O primeiro ambiente de teste foi o simulador *Mininet*. Utilizou-se a configuração padrão do *Mininet* disponível no site do projeto. Com a finalidade de facilitata a reprodutibilidade do experimento optou-se pela utilização do *Mininet* dentro de uma máquina virtual disponibilizada pela própia equipe desenvolvedora do *Mininet*.

A imagem que foi utilizada para a virtualização não sofreu edição prévia antes ou durante os testes. A configuração da máquina virtual está disponível na tabela 2.3.

Conforme já citado, o *Mininet* utiliza um sistema de arquivos compartilhado entre todos os seus *hosts* virtuais. No caso do *benchmark* de cópia de dados, o arquivo foi lido e escrito para um mesmo disco, situação na qual este processo pode agir como um *bootleneck* no *benchmark*.

| Aspecto | Medida |
|---|---|
| Número de Nucleos da CPU | 1 |
| Arquitetura | x86_64 |
| CPU | Intel(R) Core(TM) i5-3570K |
| CPU Clock | 3430 MHz |
| RAM disponível | 2000 Megabytes |
| SWAP disponível | 1024 Megabytes |
| SO virtualizado | Ubuntu 14.04 Trusty Tahr LTS |

Tabela 2.2: Configuração da máquina virtual para o teste *Mininet*

**Ambiente de simulação da rede tradicional virtualizado**

A segundo ambiente de testes foi uma rede tradiconal virtualizada. Por rede tradicional entende-se como rede onde não ocorre a separação entre o *control plane* e *data plane*. Optou-se pela utilização dos recursos de virtualização com o objetivo de ter um terceiro item para utilizar como comparativo.

As redes foram virtualizadas em uma nuvem baseada em *OpenStack*. Utilizou-se duas máquinas virtuais operando com configurações idênticas. As redes foram construídas utilizando os recursos de virtualização de rede do *OpenStack* assim como os roteadores virtuais para interligar as redes.

| Aspecto | Medida |
|---|---|
| Número de Nucleos da CPU | 1 |
| Arquitetura | x86_64 |
| CPU | Intel(R) Xeon(TM) E312xx |
| CPU Clock | 2100 MHz |
| RAM disponível | 8000 Megabytes |
| SWAP disponível | 8192 Megabytes |
| SO virtualizado | Ubuntu 14.10 Utopic Unicorn |

Tabela 2.3: Configuração da máquina virtual para o teste de rede tradicional

**Ambiente de simulação da rede tradicional não virtualizado**

O terceiro ambiente de testes foram máquinas físicas operando em uma rede de produção, algo próximo de uma situação real. Ambos os *hosts* são máquinas idênticas com configurações conforme a tabela 2.4. De maneira semelhante ao ambiente do *Mininet*, o ambiente físico também possui um sistema de arquivos compartilhado entre os *hosts*. O compartilhamento é feito via rede através do Network File System (NFS). Se por um lado isso possui um impacto negativo no desempenho, por outro se aproxima do ambiente do *Mininet*. Esta configuração foi escolhida por motivos de acesso e disponibilidade de *hardware*.

| Aspecto | Medida |
|---|---|
| Número de Núcleos da CPU | 4 |
| Arquitetura | x86_64 |
| CPU | AMD Phenom(tm) II X4 B93 Processor |
| CPU Clock | 2800 MHz |
| RAM disponível | 4000 Megabytes |
| SWAP disponível | 4096 Megabytes |
| SO virtualizado | Linux Mint 17 Quiana |

Tabela 2.4: Configuração da máquina física para o teste de rede tradicional

### 2.2.4 Experimentos

Esta subseção é dedicada à apresentação dos resultados dos testes realizados. A análise dos resultados é realizada na seção seguinte. Os *benchmarks* relativos à taxa de transferências foram executados cada um de maneira automática fazendo o uso de *Shell Script*.

Nos gráficos das figuras 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15 e 2.16, está representado no eixo x o tempo de transferencia em segundos e no eixo y a taxa de transferência em Kilobytes por segundo. A curva em vermelho representa os dados crus, enquanto que a curva em verde representa os dados suavizados, com o objetivo de eliminar ruídos visuais e facilitar a interpretação.

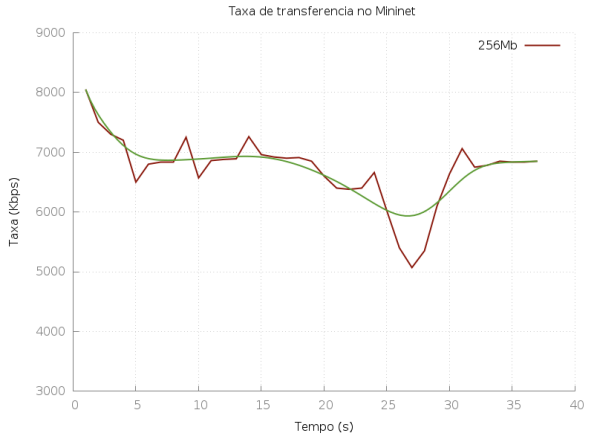


Figura 2.5: Taxa de transferência de arquivo de 256 Megabytes no Mininet

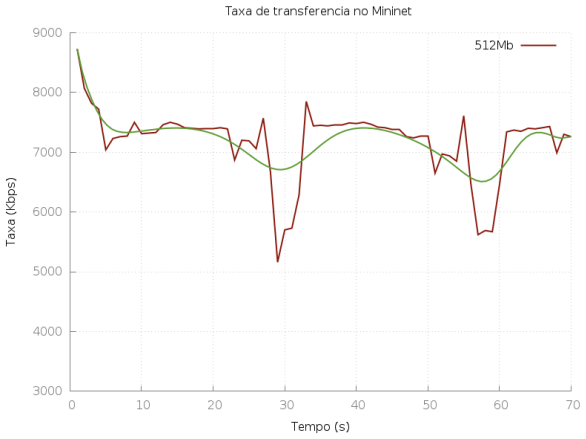


Figura 2.6: Taxa de transferência de arquivo de 512 Megabytes no Mininet

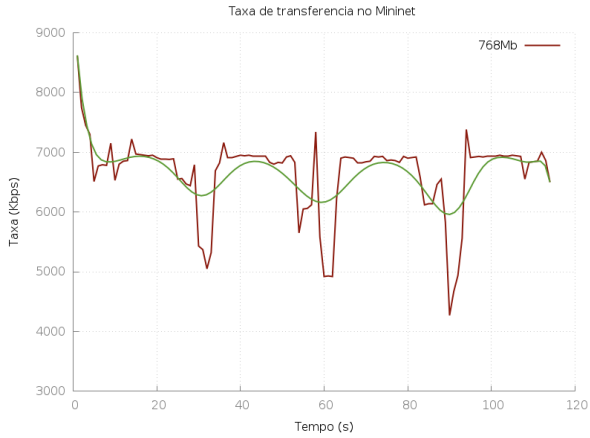


Figura 2.7: Taxa de transferência de arquivo de 768 Megabytes no Mininet


Figura 2.8: Taxa de transferência de arquivo de 1024 Megabytes no Mininet

### 2.2.5 Análise dos resultados

#### Análise da taxa de transferência

Ao se analisar as taxas de transferências do *OpenStack* notou-se um desempenho significativamente menor do que o esperado. Apesar da causa não ter sido encontrada durante o desenvolvimento deste trabalho, especula-se que o motivo é um limite arbitrário na velocidade da rede configurado pelos administradores da nuvem.

Observou-se uma taxa de transferência média conforme a tabela 2.5. O desempenho inferior do *OpenStack* é visível, portanto seus valores serão desconsiderados para este trabalho, já que não se pode identificar a causa da perda de performance para se obter o esperado em uma rede física. Utilizar-se-á portanto os resultados da rede real


Figura 2.9: Taxa de transferência de arquivo de 256 Megabytes no OpenStack


Figura 2.10: Taxa de transferência de arquivo de 512 Megabytes no OpenStack

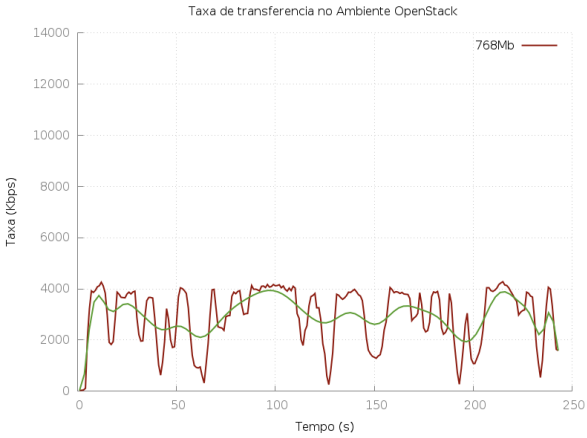


Figura 2.11: Taxa de transferência de arquivo de 768 Megabytes no OpenStack

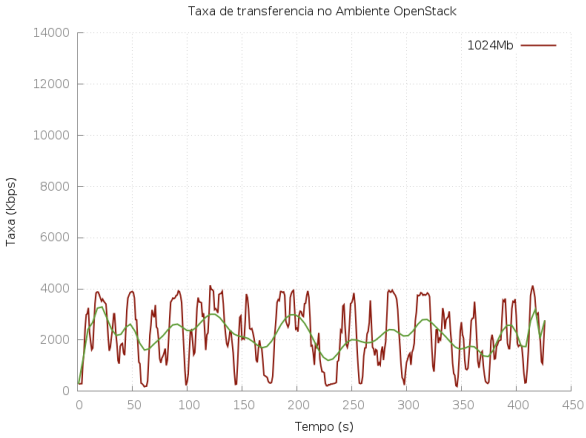


Figura 2.12: Taxa de transferência de arquivo de 1024 Megabytes no OpenStack

como comparativo ao desempenho do *Mininet*.

A primeira coisa que se notou foi o fato de que as transferências no *Mininet* foram levemente superiores a do ambiente físico. O *Mininet* manteve uma taxa de transferência média de 6.90 MB/s nos testes realizados, enquanto que a rede física manteve uma taxa de 6.625 MB/s. Uma diferença de aproximadamente 4.25%. Essa diferença por si só não é conclusiva.

Observando-se o desvio padrão nota-se que o ambiente real obteve um valor de aproximadamente uma ordem de magnitude maior quando comparado ao *Mininet*. Isso é facilmente explicado pelo fato de que a rede simulada no *Mininet* é uma rede isolada onde apenas os *benchmarks* estavam consumindo recursos computacionais significativos. Já para a rede física, os testes foram realizados em uma rede de produção, onde ocorria

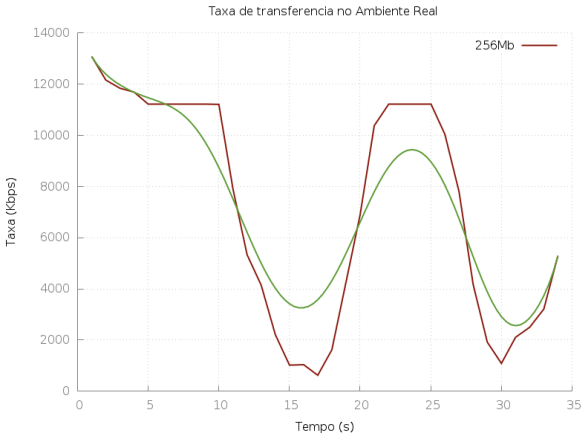


Figura 2.13: Taxa de transferência de arquivo de 256 Megabytes no ambiente físico

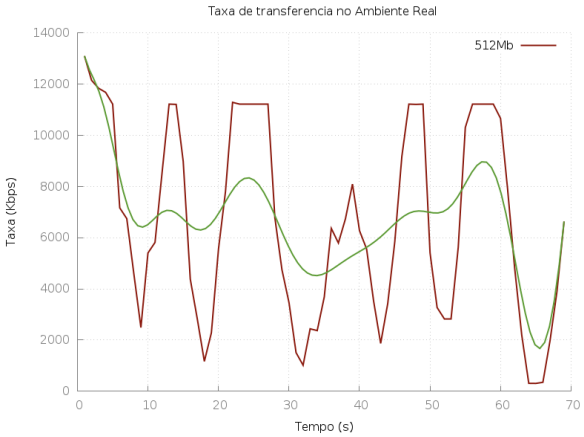


Figura 2.14: Taxa de transferência de arquivo de 512 Megabytes no ambiente físico

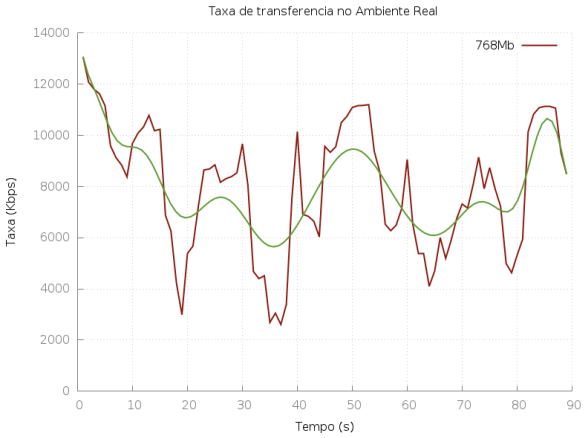


Figura 2.15: Taxa de transferência de arquivo de 768 Megabytes no ambiente físico

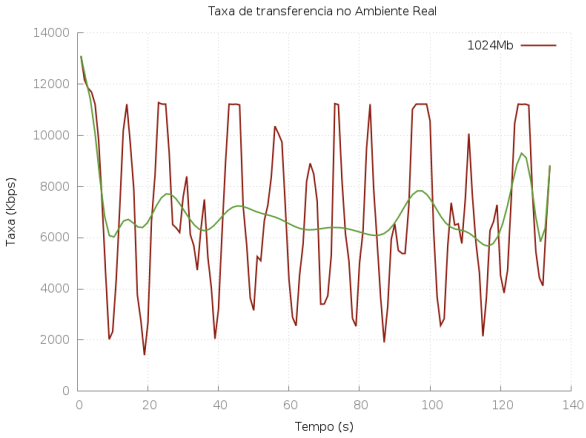


Figura 2.16: Taxa de transferência de arquivo de 1024 Megabytes no ambiente físico

uma demanda maior por recursos computacionais. Outro aspecto interessante é o fato de que no *Mininet* a taxa de transferência não se aproximou do seu limite teórico de 100 Mbits/s, porém se manteve estável próximo do valor médio de taxa de transmissão, o que é visível baseado no desvio padrão relativamente baixo quando comparado ao do ambiente real. O ambiente físico, por sua vez, chegou diversas vezes próximo do seu limite teórico, porém teve também quedas abruptas nas quais ficou abaixo da taxa de transmissão de 2 MB/s.

Nas figuras 2.17 e 2.18 tem-se um gráfico comparativo lado a lado entre o ambiente simulado do *Mininet* e o ambiente físico. Uma simples comparação visual confirma o que se concluiu no paragrafo anterior. Na figura 2.18 é visível a maior média das taxas de transferências e também uma menor variação entre as taxas para diferentes casos. Já

| Tamanho do arquivo | Real | Mininet | OpenStack |
|---|---|---|---|
| 256 Megabytes | 6.17 MB/s | 6.67 MB/s | 2.57 MB/s |
| 512 Megabytes | 6.07 MB/s | 7.15 MB/s | 2.26 MB/s |
| 768 Megabytes | 7.62 MB/s | 6.61 MB/s | 2.36 MB/s |
| 1024 Megabytes | 6.64 MB/s | 7.17 MB/s | 2.10 MB/s |

Tabela 2.5: Taxa de transferência média

| Tamanho do arquivo | Real | Mininet |
|---|---|---|
| 256 Megabytes | 4.26 | 0.560 |
| 512 Megabytes | 3.76 | 0.594 |
| 768 Megabytes | 2.47 | 0.645 |
| 1024 Megabytes | 2.95 | 0.549 |

Tabela 2.6: Desvio padrão da taxa de transferência

para o ambiente físico, apresentado na figura 2.17, a dispersão das taxas de transferências é evidente. É possível notar que entre uma transferência e outra houve uma variação de aproximadamente 1.5 MB/s.

Apesar do desempenho superior do *Mininet*, é plausível especular que o fato da rede física utilizada para os testes não estar isolada afetou os resultados negativamente. Foram observados taxas de transmissões no ambiente físico próximas do limite teórico da rede, o que indica que o desempenho poderia ter sido superior caso não fosse pelas bruscas quedas de velocidade de transferência. Existem suspeitas de que o desempenho do *Mininet* poderia ter sido superior ao registrado, levando em conta que o ambiente foi virtualizado e que os *hosts virtuais* estavam utilizando um sistema de arquivos compartilhado. Porém, por motivos de reprodutibilidade, optou-se por utilizar o ambiente virtualizado que funciona *out of the box*.

Com o objetivo de eliminar interferências externas, os testes foram realizados múltiplas vezes. Na grande maioria dos casos os resultados foram iguais. No caso do ambiente de rede física houve um único teste na qual a taxa de transferência ficou abaixo de 2 MB/s. Esse resultado fpra descartado já que se considerou que algum fator externo influenciou a rede do ambiente de testes ou alguma das máquinas onde se realizou os testes. No *Mininet* observou-se também um comportamento anormal diversas vezes. Em diversos testes a velocidade de transferência subtamente caía para aproximadamente 500

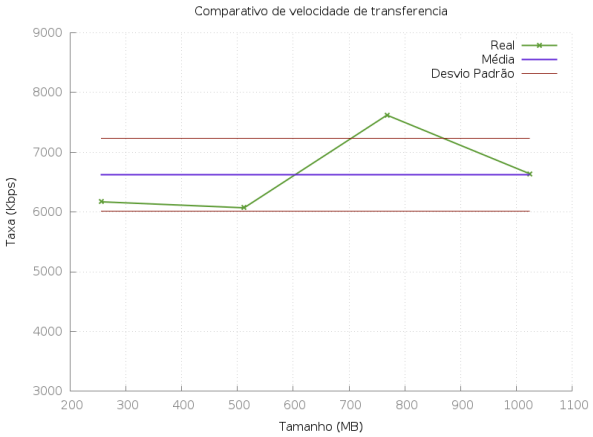


Figura 2.17: Taxa, média e desvio padrão de transferência no ambiente físico

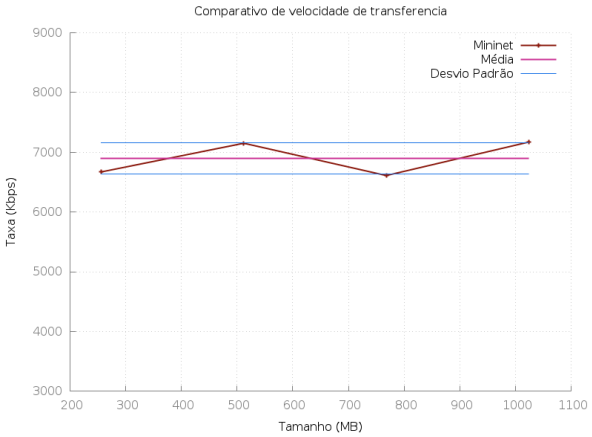


Figura 2.18: Taxa, média e desvio padrão de transferência no ambiente Mininet

KB/s e se mantinha constante neste valor até o término da simulação. Na figura 2.19 é possível visualizar este comportamento. Após aproximadamente 2 minutos de transferência ocorreu a queda de velocidade. Nota-se que a taxa se mantem aproximadamente constante. Durante esse periodo de anormalidade, a taxa de transferência média foi de 495.2 KB/s e a variação média foi de apenas 29.6 KB/s. Em diversas ocorrências desse fenômeno foram encontrados valores semelhantes. Não encontrou-se a causa deste comportamento. Testes com o *iperf* demonstraram que a rede estava configurada conforme o esperado. Também notou-se que não ocorreram quedas significativas de desempenho ao utilizar-se o *iperf*.

### Análise do tempo de inicialização

Analisando-se a figura 2.20 tem-se os resultados dos testes de consumo de Random Access Memory (RAM) pelo *Mininet*. Nota-se que o consumo de RAM para a simulação de pequenas redes facilmente ultrapassa a marca dos 400 Megabytes. Isto porém não se apresenta como um problema quando levado em comparação a alta disponibilidade de RAM que se tem nos computadores modernos. No entanto, para topologias maiores, chegou-se ao limite de RAM disponível no sistema ao se ter uma topologia do tipo Linear com 800 *hosts* e 800 roteadores simulados. O teste seguiu até se obter 1024 roteadores e 1024 *hosts* na rede, apesar da utilização de *SWAP*, não houve perdas consideráveis no tempo de inicialização, conforme mostra a figura 2.21. Para a rede de topologia do tipo *Single*, atingiu-se a marca de 1024 *hosts* virtuais sem esgotar os recursos da máquina, isso é facilmente explicado pelo fato de haver apenas um único roteador na infraestrutura

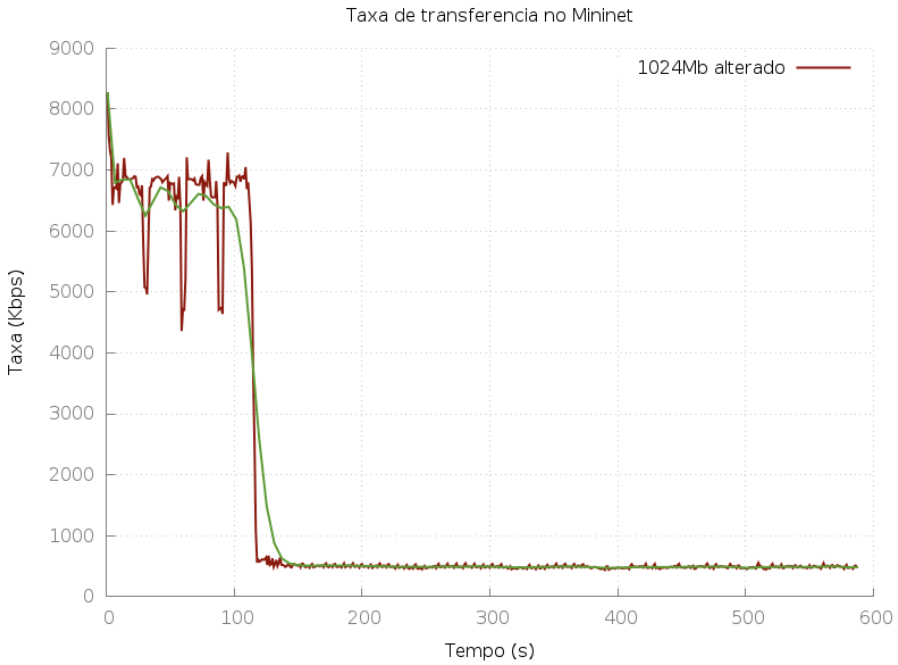


Figura 2.19: Comportamento anormal no Mininet

simulada.

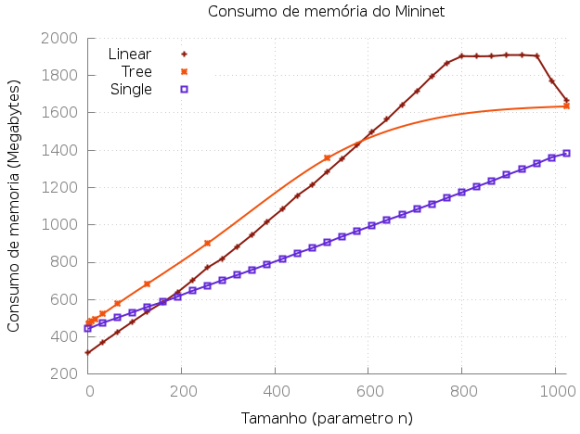


Figura 2.20: Consumo de RAM pelo Mininet

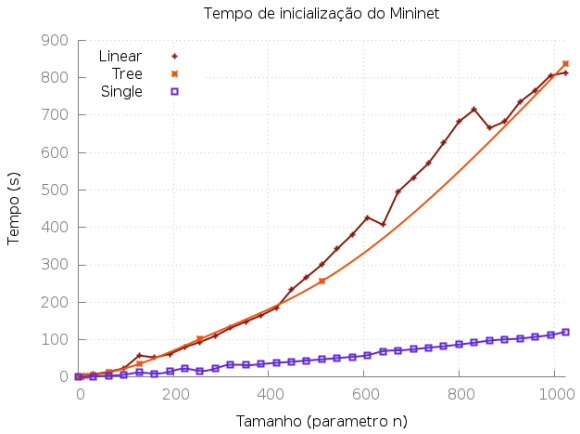


Figura 2.21: Tempos de boot do Mininet

Relativo ao tempo necessário para a inicialização do *Mininet*, nota-se que a topologia do tipo *Single* possui um custo consideravelmente baixo quando comparado com as demais topologias testadas. Isto ocorre pelo fato de que a topologia em questão é constituida de um único *switch* e múltiplos *hosts* conectados ao *switch*. Já para as topologias linear e em árvore, nota-se um crescimento polinomial no tempo necessário para se inicializar o *Mininet*. Essa diferença se deve provalmente a um custo maior em criar *switches* virtuais e os *links* do que os *hosts*.

## 2.3 Conclusão

Durante este trabalho atingiu-se parcialmente o seu objetivo de análisar e comparar o desempenho do *Mininet*. O *Mininet* obteve um desempenho médio 4.45% superior ao ambiente físico e uma variação média de apenas 17% da variação média do ambiente físico. No entanto não foi possível determinar a causa de algumas anomalias detectadas. No caso do *Mininet* não foi possível isolar a causa da súbita queda de performance. Para o cenário do *OpenStack* optou-se por descartar os resultados obtidos, tendo em vista que os dados apresentaram um limite superior na taxa de transmissão, que se mostrou inadequada para a comparação com os demais ambientes. Nos testes do ambiente físico não foi possível determinar se algum fator externo os influenciou, tendo em vista que a taxa de transferência chegou perto de seu limite teórico varias vezes porém sofreu múltiplas quedas de velocidade. Não obstante, o trabalho cumpriu seus objetivos, mesmo que de maneira parcial, e foram obtidos os resultados esperados. O *Mininet* mostrou-se uma ferramenta capaz de cumprir seus objetivos e teve um comportamento equiparável com os resultados obtidos dos testes em uma rede física.

# Referências Bibliográficas


ANTONENKO, V.; SMELYANSKIY, R. Global network modelling based on mininet approach. 2013.

BASH Reference Manual. 2015. `https://www.gnu.org/software/bash/manual/bash.html`. [Online; accessed 25-Outubro-2015].

BSD License. 2015. `https://opensource.org/licenses/BSD-3-Clause`. [Online; accessed 25-Outubro-2015].

FEAMSTER, J. R. N.; ZEGURA, E. *The road to SDN: An intellectual history of programmable networks*. [S.l.: s.n.], 2011.

FREE(1) - Linux manual page. 2015. `http://linux.die.net/man/1/free`. [Online; accessed 25-Outubro-2015].

HANDIGOL, N. et al. Reproducible network experiments using container-based emulation. p. 253–264, 2012.

IP-NETNS - Linux manual page. 2015. `http://man7.org/linux/man-pages/man8/ip-netns.8.html`. [Online; accessed 25-Outubro-2015].

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. *CoRR*, 2014.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop, rapid prototyping for software-defined networks. 2010.

LINUX Kernel Documentation. 2015. `https://www.kernel.org/doc/Documentation/sysctl/vm.txt`. [Online; accessed 25-Outubro-2015].

LINUX Programmer's Manual, slab info. 2015. `http://man7.org/linux/man-pages/man5/slabinfo.5.html`. [Online; accessed 25-Outubro-2015].

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MININET License. 2015. https://github.com/mininet/mininet/blob/master/LICENSE. [Online; accessed 25-Outubro-2015].

MININET Python API Reference Manual. 2015. http://mininet.org/api/index.html. [Online; accessed 28-Outubro-2015].

OLIVEIRA, R. L. S. de et al. Using mininet for emulation and prototyping software-defined networks. 2013.

ONF. *Software-Defined Networking: The New Norm for Networks*. [S.l.: s.n.], 2012.

ONF. *SDN Architecture Overview*. [S.l.: s.n.], 2013.

ONF. *SDN Architecture*. [S.l.: s.n.], 2014.

ONF. *SDN in the Campus Environment*. [S.l.: s.n.], 2014.

RSYNC(1) - Linux manual page. 2015. http://linux.die.net/man/1/rsync. [Online; accessed 18-Novembro-2015].

WETTE, P. et al. Maxinet: Distributed emulation of software-defined networks. June 2014.