CRIAÇÃO E COLETA DE DADOS ESTATÍSTICOS EM REDES DEFINIDAS POR *SOFTWARE* (SDN)

Trabalho de Conclusão de Curso

Engenharia da Computação

José Alves Muniz Neto
Orientador: Prof. Edison de Queiroz Albuquerque

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

JOSÉ ALVES MUNIZ NETO

CRIAÇÃO E COLETA DE DADOS ESTATÍSTICOS EM REDES DEFINIDAS POR *SOFTWARE* (SDN)

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, dezembro 2018.

Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação
Projeto de Final de Curso

ESCOLA POLITÉCNICA
DE PERNAMBUCO

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 28 de dezembro de 2018, às 9:00 horas, reuniu-se para deliberar a defesa da monografia de conclusão de curso do discente **JOSE ALVES MUNIZ NETO**, orientado pelo professor **Edison de Queiroz Albuquerque**, sob título **Criação e coleta de dados estatísticos em redes definidas por software (SDN)**, a banca composta pelos professores:

Sérgio Murilo Maciel Fernandes
Edison de Queiroz Albuquerque

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

☐ Aprovada ☒ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 9,0 (NOVE)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O discente terá 30 dias para entrega da versão final da monografia a contar da data deste documento.

SÉRGIO MURILO MACIEL FERNANDES

EDISON DE QUEIROZ ALBUQUERQUE

* Este documento deverá ser encadernado juntamente com a monografia em versão final.

Agradecimentos

Primeiramente, agradeço a Deus por ter me guiado até aqui e ter me protegido em toda a minha trajetória, essa que abençoou com saúde, paz e amor.

Ao meu pai, José Alves Muniz Junior, gostaria de agradecer por ser meu guia e inspiração de pessoa hoje e para sempre, este que não mediu esforços para me ajudar nesta caminhada, me proporcionando inúmeras oportunidades de experiências e novos conhecimentos para que eu conseguisse chegar até aqui. Quero agradecer também a minha mãe Carla Fernanda de Almeida Monteiro Muniz, que junto ao meu pai, sempre me apoiou e me deu forças para nunca desistir, me dando coragem para sempre chegar ao topo. Quero que saibam que toda dedicação realizada por eles na minha educação foi de suma importância para minha vida diária.

Em especial, quero agradecer também a minha noiva, amiga e companheira Camila Drummond, por todo amor, carinho, compreensão e paciência cedida para que eu pudesse realizar este trabalho e esta formação.

Quero lembrar também as amizades que a universidade me permitiu criar, levarei todos em meu coração e espero que na minha vida profissional também.

Por fim, mas não menos importante, devo esta dedicação aos grandes professores desta universidade, principalmente ao meu professor orientador Edison de Queiroz, que sem eles esta construção profissional não seria possível, muito obrigado!

Resumo

Com o passar dos anos a tecnologia da informação vem evoluindo e crescendo de uma forma astronômica, cada vez mais serviços estão sendo incorporados às novas tecnologias, tais como voz e vídeo. Devido a este crescimento, fica cada vez mais comum o aumento na largura de banda, a complexidade dos protocolos e as funções tradicionais das redes de computadores. Consequentemente, estes eventos proporcionam o surgimento de diversos problemas, como baixa performance e interrupções nos serviços, que se tornam cada vez mais significantes e responsáveis por perdas financeiras e de credibilidade entre os consumidores. Com o intuito de solucionar tais problemas, surge o conceito de Redes Definidas por *Software* (SDN, do inglês *Software Defined Network*), visando melhorar a monitoração de operações e atender a intensificação do volume de dados. Este documento propõe um modelo elaborado de um controlador de rede, utilizando o protocolo *OpenFlow*, que determina instruções de encaminhamento de pacotes de dados recebidos pelos switches da rede, e também apresenta um modelo de uma função para o monitoramento e coleta de dados estatísticos da rede criada.

Palavras-chave: Tecnologia da Informação, Largura de Banda, Protocolos, Redes de Computadores, Redes Definidas por *Software*, Dados, Controlador, *OpenFlow*, Pacotes, Funcão, Monitoramento, Coleta de Dados.

Abstract

Over the years, information technology has been evolving and growing in an astronomical way, and more and more services are being incorporated into new technologies, such as voice and video. Due to this growth, the increase in bandwidth and complexity of the traditional protocols and functions of computer networks becomes increasingly common. Consequently, such events give rise to a number of problems, such as poor performance and interruptions in services, which become increasingly significant and responsible for financial losses and credibility among consumers. In order to solve these problems, the concept of Software Defined Network (SDN) arises to improve the monitoring of operations and to attend to the intensification of data volume. This paper proposes an elaborate model of a network controller, using the OpenFlow protocol, which determines routing instructions for packets of data received by the network switches, and also presents a model of a function for the monitoring and data collection of the network created.

Keywords: Information Technology, Bandwidth, Protocols, Computer Networks, Software Defined Networks, Data, Controller, OpenFlow, Packets, Function, Monitoring, Data Collection.

Sumário

Índice de Figuras

Tabela de Símbolos e Siglas

(Dispostos em ordem alfabética)

API – *Application Programming Interface* (Interface de Programação de Aplicativos);

ATM – *Asynchronous Transfer Mode* (Modo de Transferência Assíncrona);

BPS – *Bits per second* (Bits por segundo);

BWC – *BandWidth Controller* (Controlador de Largura de Banda);

CPU – *Central Process Unit* (Unidade Central de Processamento);

Host – Cliente utilizador dos serviços do servidor;

Layer – Camada de função do modelo OSI;

ONF - *Open Networking Foudation*;

OSI – *Open Systems Interconnection;*

P2P – *Peer to Peer*;

QoS – *Quality of Service* (Qualidade de serviço);

Queue – Fila;

SDH – *Synchronous Digital Hierarchy* (Hierarquia Digital Síncrona);

SDN – *Sofware Defined Network* (Redes Definidas por *Software*);

Software – Conjunto de componentes lógicos de um computador ou sistema de processamento de dados;

VM – *Virtual Machine* (Máquina Virtual);

1 Introdução

Neste documento de monografia foi utilizado um algoritmo computacional baseado em encaminhamento e processamento de dados, coletados a partir de uma rede virtual criada. Utilizando tecnologias conhecidas como Redes Definidas por *Software*, visando transformar os *switches* tradicionais em *switches* inteligentes, com aprendizado, reescrita e redirecionamento de endereços de rede dos pacotes, a partir de instruções enviadas pelo controlador ao *switch*. Com isso, é possível a realizar comutação entre as portas de saída de um *switch*. O trabalho proposto utiliza ferramentas de criação de ambientes virtuais e tráfego de dados para a realização de testes de eficiência.

Este capítulo apresenta a introdução deste trabalho de conclusão de curso, estruturado em três subseções. Primeiramente é apresentada a motivação para a produção desta pesquisa e também o problema a ser tratado, em seguida são apresentados os objetivos gerais e a solução proposta para o problema encontrado. Por fim, é relatado a organização restante do trabalho.

1.1 Motivação e Caracterização do Problema

Atualmente, redes de telecomunicações estão cada vez mais complexas por conta da constante evolução na área de tecnologia da informação. Devido a este contínuo crescimento, cada vez mais serviços estão sendo incorporados às novas tecnologias e progressivamente o aumento na largura de banda e na complexidade dos protocolos e funções tradicionais se tornam cada vez mais comuns. Consequentemente, tal desenvolvimento ocasiona o aparecimento de diversos problemas, tais como baixa performance e interrupções nos serviços.

Visando uma boa qualidade de serviço (*QoS – Quality of Service*) na rede, é necessário o controle do tráfego de pacotes na rede. Com essa finalidade, uma SDN, incorporada com o protocolo *OpenFlow*, tem conseguido uma melhoria na comunicação do plano de controle com o plano de dados da rede, além de suprimir a rigidez presente nas redes tradicionais.

1.2 Objetivos e Metas

Por conta do grande crescimento das redes de computadores ao longo do anos, os servidores de internet estão sendo cada vez mais requisitados para a execução de serviços. Devido a este fato, este trabalho tem como principal objetivo a utilização de dois algoritmos computacionais: um para a criação de um controlador, que será responsável por determinar instruções de encaminhamento a respeito dos pacotes recebidos pelo *switch*; outro, que utiliza o conceito de herança, de programação orientada a objetos, que será utilizado para a monitoração e coleta de dados estatísticos do *switch* utilizado, a fim de detectar erros e suas causas. Será utilizada a plataforma de emulação de rede, *Mininet*, para a criação de um ambiente virtual a ser usado.

1.3 Organização do Documento

Este documento está estruturado em 5 capítulos. No Capítulo 2, serão revistos alguns fundamentos que foram explorados na pesquisa realizada para o desenvolvimento deste projeto, começando com conceitos e definições que foram utilizados no desenvolvimento do projeto. No Capítulo 3 serão apresentados os materiais e ferramentas utilizados para o desenvolvimento do algoritmo elaborado, e em seguida é demonstrada a metodologia utilizada para a realização da pesquisa. Já no Capítulo 4, serão demonstrados os testes e resultados obtidos a partir da utilização dos algoritmos. Finalmente, no Capítulo 5, serão debatidas as conclusões sobre este trabalho, juntamente com os possíveis trabalhos futuros a partir deste projeto.

2 Fundamentação Teórica

2.1 Redes Definidas por *Software* (SDN)

Surgido em 2005, o conceito de redes definidas por *software* revela-se como um novo paradigma de redes programáveis, afim de melhorar a monitoração de operações e atender a intensificação do volume de dados. Uma rede definida por *software* é caracterizada pela existência de um sistema de controle (controlador) que pode controlar o mecanismo de encaminhamento dos elementos de comutação da rede por uma interface de programação bem definida [3].

Mais especificamente, a ideia principal de uma SDN é separar o controle da rede da transmissão de dados, sendo este controle diretamente programável, ou seja, todo o gerenciamento da rede se torna programável mediante um controlador. Com essa separação, o controlador possui uma visão global de toda a rede e toma decisões em tempo real. Em outras palavras, no paradigma de uma SDN, nem todos os processos ocorrem dentro do mesmo dispositivo. Com isso, o *hardware* (*switch*) passa a ter apenas uma tarefa, encaminhar os pacotes de dados de acordo com as regras que são determinadas pelo controlador da rede.

A estrutura de uma SDN, representada na Figura 1, é repartida em três camadas. A camada mais abaixo é a camada de infraestrutura, também chamada de plano de dados ou de encaminhamento, constituída pelos elementos de encaminhamento de rede, tendo como principal objetivo o encaminhamento de dados, bem como o monitoramento de informações locais e a coleta de estatísticas da rede, armazenar os dados coletados temporariamente em dispositivos locais e passá-los ao controlador [6].

A camada do meio é a camada de controle, também chamada de plano de controle, que age como uma interface entre a camada de infraestrutura e a camada de aplicação. É responsável por programar e gerenciar a camada de infraestrutura.

Para isso, utiliza as informações fornecidas pelo plano de dados e define a operação e o roteamento da rede. Este plano consistem em um ou mais controladores de rede que se comunicam com os elementos da rede de encaminhamento, por meio de interfaces padronizadas que são chamadas de interfaces para o sul [6].

A camada mais acima, é a camada de aplicação, que se resume a uma aplicação SDN projetada para atender ao requisito feito pelo usuário. Esta contém aplicativos de rede que podem introduzir novos recursos de rede, como segurança e gerenciamento, esquemas de encaminhamento ou ainda auxiliar a camada de controle na configuração da rede. A camada de aplicação pode receber uma visão abstrata e global da rede de controladores e usar essa informação para fornecer orientação apropriada à camada de controle [6].

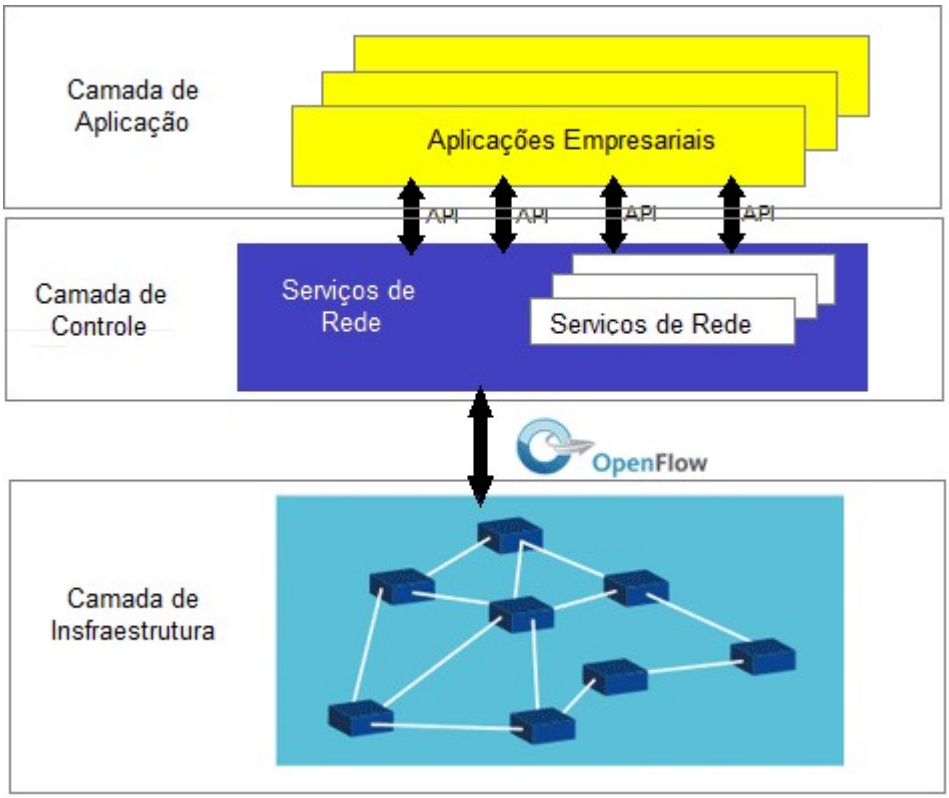

Figura 1. Divisão das camadas de uma SDN.

2.1.1 OpenFlow

OpenFlow é um protocolo-padrão de comunicação para determinar as ações de encaminhamento de pacotes em dispositivos de rede, como, por exemplo,

comutadores, roteadores e pontos de acesso sem fio [7]. Neste sentido, a contribuição mais importante do paradigma do *OpenFlow* é a generalização do plano de dados, que de forma pragmática reutiliza as funcionalidades do *hardware* existente através da definição de um conjunto simples de regras, ou qualquer modelo de encaminhamento de dados baseado na tomada de decisão fundamentada em algum valor ou combinação de valores dos campos de cabeçalho dos pacotes, e das ações associadas: encaminhar, descartar, enviar para o controlador, reescrever campos do cabeçalho, etc.

O protocolo OpenFlow é o protocolo mais utilizado para a interface sul da SDN, que separa o plano de dados do plano de controle. O OpenFlow foi inicialmente proposto pela Universidade de *Stanford*, para atender à demanda de validação de novas propostas de arquiteturas e protocolos de rede sobre equipamentos comerciais, e agora é padronizado pelo ONF (*Open Networking Foudation*) [7].

A arquitetura OpenFlow consiste em três conceitos básicos:

1. A rede é construída por switches compatíveis com OpenFlow que compõem o plano de dados;
2. O plano de controle consiste em um ou mais controladores OpenFlow;
3. Um canal de controle seguro conecta os comutadores ao plano de controle.

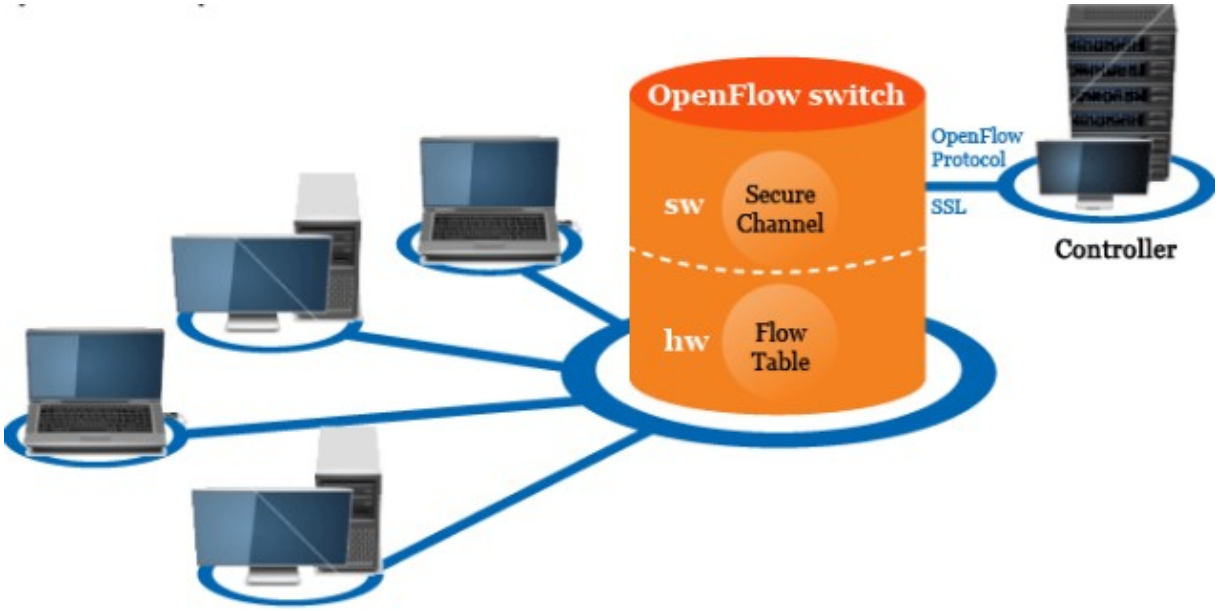

Figura 2. Exemplo de uma rede com a utilização do protocolo *OpenFlow*.

2.1.2 Controlador SDN

A arquitetura SDN não especifica o design interno ou a implementação de um controlador SDN. Este poderia ser um processo monolítico único, poderia ser uma confederação de processos idênticos organizados para compartilhar carga ou proteger uns aos outros de falhas, poderia ser um conjunto de componentes funcionais distintos em um arranjo colaborativo, poderia assinar serviços externos para algumas de suas funções, como por exemplo, computação de caminho. Qualquer combinação dessas alternativas é permitida: o controlador SDN é visto como uma caixa preta, definida por seu comportamento externamente observável [11].

O controlador de uma rede SDN é desenvolvido para ter uma visão global de toda a topologia da rede e que seus componentes sejam determinados para compartilhar informações de tal forma que nenhum bloco externo precise se preocupar com comandos conflitantes ou contraditórios por parte do controlador.

Vários gerenciadores ou componentes do controlador podem ter acesso conjunto de gravação aos recursos da rede, mas para cumprir os princípios da SDN, eles devem:

a) ser configurado para controlar conjuntos disjuntos de recursos ou ações, ou;

b) ser sincronizados entre si para que nunca emitam comandos inconsistentes ou conflitantes.

2.2 *Quality of Service (QoS)*

QoS se refere às habilidades que um conjunto de tecnologias (*Frame Relay*, ATM, redes *Ethernet* e 802.1, SDH e redes roteadas por *IP*) pode oferecer a rede, como garantir a largura de banda suficiente para a operação, fornecer melhores serviços para uma rede selecionada, gerir de forma rentável o tráfego da rede, melhorar as experiências do usuário, etc.

Mais especificamente, os recursos de *QoS* oferecem um serviço melhor e mais previsível para:

- Suportar largura de banda dedicada;
- Melhorar as características de perda;
- Evitar e gerenciar o congestionamento de rede;
- Modelar o tráfego da rede;
- Definir as prioridades de tráfego na rede.

Controle de tráfego e gerenciamento de largura de banda são as duas principais formas para se alcançar QoS, ou seja, QoS é essencial para o controle de tráfego e largura de banda [8].

2.3 Herança

A redução de custos com *software* não está em contratar programadores baratos e mal qualificados. Isso pode prejudicar bastante o tempo de entrega do projeto e sua usabilidade, além de dificultar a manutenção do código. A redução dos

custos de um determinado *software* se dá através do reuso e adaptabilidade do código. Uma das formas de reuso de *software* é a herança. [17].

A herança é um mecanismo que permite que uma nova classe consiga herdar propriedades de uma outra classe já existente. Quando uma classe herda da outra temos uma relação de dependência entre as duas. A nova classe a ser criada é chamada de subclasse e a classe existente que será utilizada para descrever a nova classe é chamada de superclasse. Temos que, a subclasse herda os métodos e variáveis de instância da superclasse. A subclasse é capaz de adicionar novos métodos e variáveis de instância à superclasse ou alterá-los de maneira a atender as necessidades do programador. Então para definir uma subclasse, dizemos em que ela difere da superclasse [18].

Com esse recurso é possível aproveitar uma estrutura existente de uma classe e replicar em outra. Dessa forma, é possível reutilizar códigos, sem a necessidade de reescrever ou duplicar trechos do programa.

A herança não é limitada a apenas um nível. A árvore de herança, ou hierarquia de classes, pode ser tão profunda quanto for necessário. Métodos e variáveis internas são herdados por todos os objetos dos níveis para baixo. Quanto mais para baixo na hierarquia uma classe estiver, mais especializado o seu comportamento. Várias subclasses descendentes podem herdar as características de uma superclasse ancestral [19].

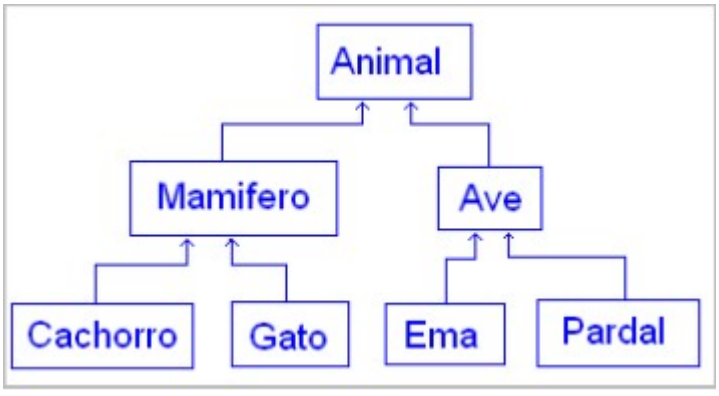

Figura 3. Exemplo de hierarquia de classes para herança.

No exemplo acima Cachorro herda diretamente de Mamífero, mas também herda indiretamente de animal, haja em vista que a herança é transitiva.

3 Desenvolvimento

O desenvolvimento do algoritmo proposto está baseado na utilização do protocolo *OpenFlow* para a criação de uma rede de computadores que possua um *switch* inteligente com tabela de aprendizado a respeito do endereçamento de pacotes de internet e também para a coleta de dados estatísticos da rede. Ao longo deste capítulo são descritos os materiais e ferramentas que foram aplicados desde a criação do ambiente até a realização de testes. Em seguida, é detalhada a metodologia utilizada na construção do modelo proposto e os passos para o desenvolvimento do sistema.

3.1 Materiais e Ferramentas

3.1.1 Sistema Operacional

Todo o experimento foi realizado em uma máquina virtual construída com o núcleo *Linux* de versão 4.15.0-34-*generic*, utilizando o sistema operacional *Ubuntu* na versão 18.04.1 LTS, com processador *Intel Core* I5-4210U, 1.70 *GHz* 64 *bits* e com memória de 21 *GB.*

3.1.2 Mininet

Ferramenta que possibilita a criação de uma rede virtual em uma única máquina (VM, *Cloud* ou nativa) em segundos, executando apenas um simples comando: “sudo mn”.

Neste projeto foi utilizado o Mininet na versão 2.3.0d1.

3.1.3 RYU

Ryu é uma estrutura de rede definida por *software* baseada em componentes de *software* com APIs bem definidas que facilitam aos desenvolvedores a criação de novos aplicativos de gerenciamento e controle de

rede. O Ryu suporta vários protocolos para gerenciar dispositivos de rede, como *OpenFlow*, Netconf, OF-*config*, etc.

Neste projeto, o Ryu foi utilizado para configurar o controlador da rede, para que o mesmo determine as intruções de redirecionamento dos pacotes recebidos pelo *switch*.

3.1.4 Protocolo *OpenFlow*

Neste projeto foi utilizado a versão 1.3 do protocolo *OpenFlow*, tanto na configuração do *switch* utilizado, quanto na criação do controlador.

3.2 Metodologia

O desenvolvimento do modelo proposto foi realizado utilizando a linguagem *Python* e é baseado na tecnologia de redes definidas por *software*, onde um controlador de rede irá tomar as decisões a respeito da comutação de entrada e saída de pacotes pelas portas de um *switch* com o protocolo *OpenFlow* habilitado, a partir de um algoritmo bem definido, para que não seja necessário um auxílio externo. As decisões ocorrem da seguinte forma:

1. Se o endereço MAC de destino do pacote for conhecido, o controlador decidirá para qual porta o pacote sairá;

2. Se não, todas as portas receberão o pacote, com exceção da porta em que o pacote entrou.

O passo-a-passo para a realização do modelo proposto é baseado no algoritmo utilizado para a monitoração e coleta de dados estatísticos da rede. Este algoritmo é uma subclasse do algoritmo utilizado para a implementação do controlador, ou seja, a partir do conceito de herança, a subclasse herda todos os métodos e variáveis de instância da superclasse. Sendo assim, ao executar a subclasse, uma instância da superclasse é criada e iniciada.

O passo-a-passo é repartido em quatro etapas, descritas a seguir:

1. Construção do ambiente:

   Arquitetura do ambiente utilizado no projeto, constituído por 1 controlador, 1 switch e 4 hosts.

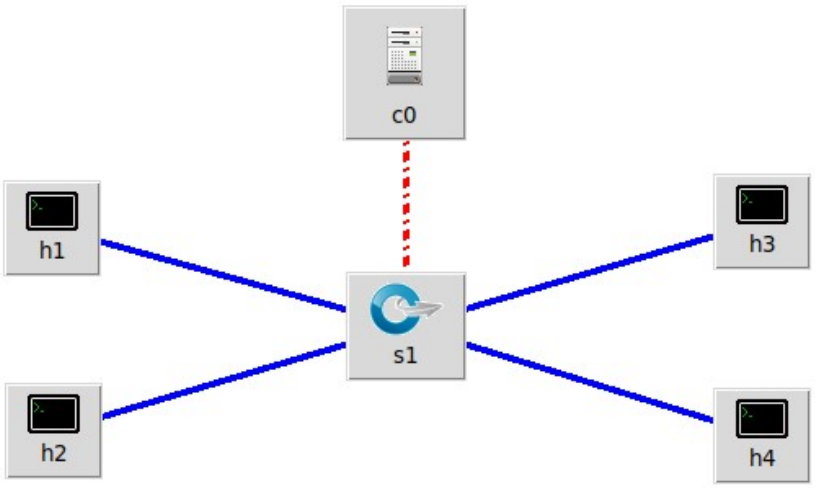

Figura 4. Topologia utilizada no experimento.

```
neto@ubuntu:~$ sudo mn --custom ~/mininet/custom/topo-1sw-4host.py --topo mytopo
 --mac --switch ovsk --controller remote
[sudo] password for neto:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Figura 5. Criação do ambiente customizado.

2. Habilitação do protocolo *OpenFlow* no *switch*:

Após a execução do comando de elaboração do ambiente, primeiramente é necessário configurar o *switch* para a habilitação do protocolo *OpenFlow*:

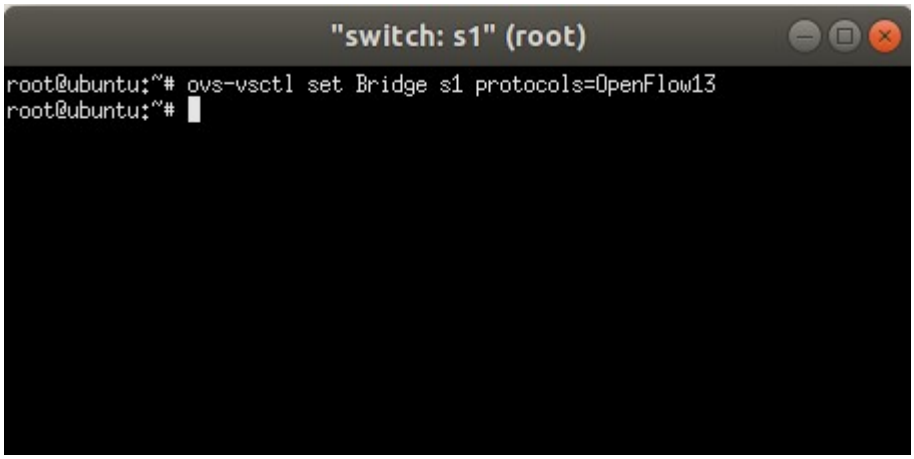

Figura 6. Habilitação do protocolo *OpenFlow* no *switch*.

3. Execução e inicialização do controlador da rede:

Nesta etapa, é inicializado o algoritmo de monitoramento e coleta de dados (*simple_monitor_13*) no terminal do controlador da rede.

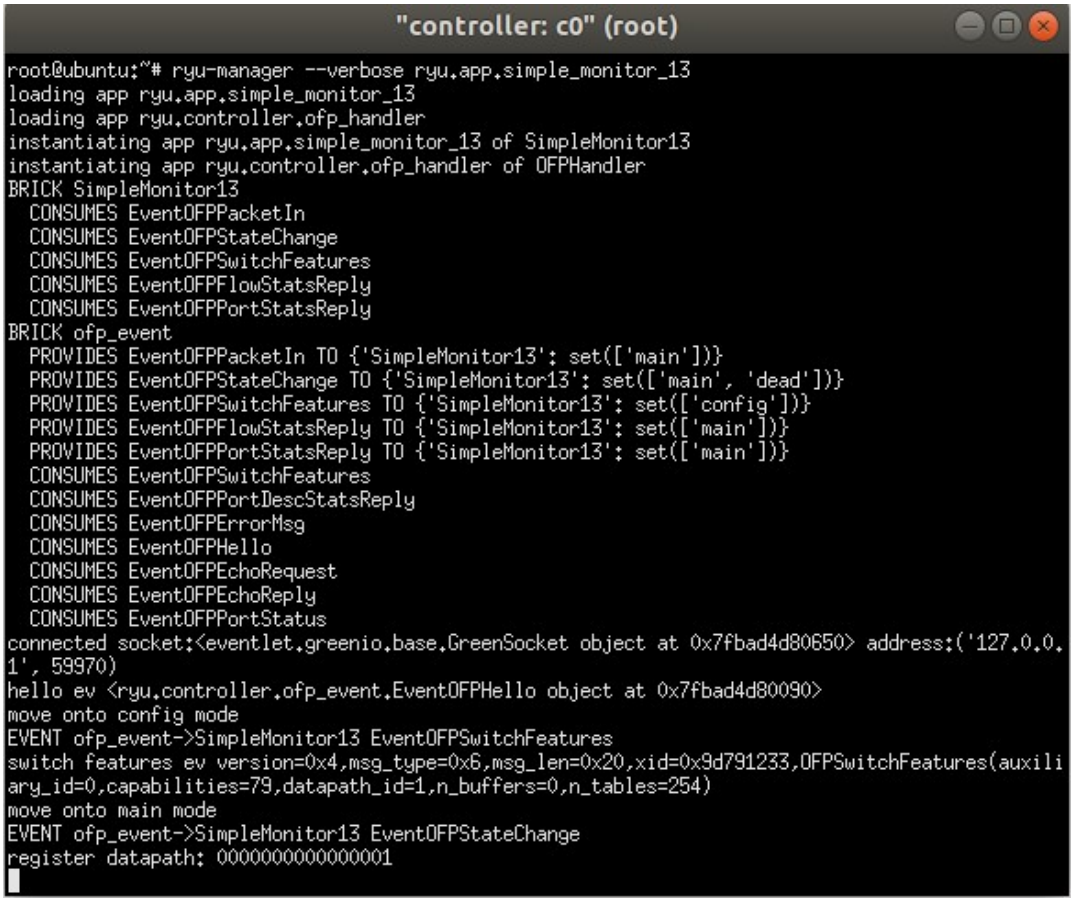

Figura 7. Inicialização do controlador da rede.

4. Comunicação entre os *hosts* para que o *switch* comece a receber os pacotes da rede e o monitoramento e coleta de dados possa ser executado:

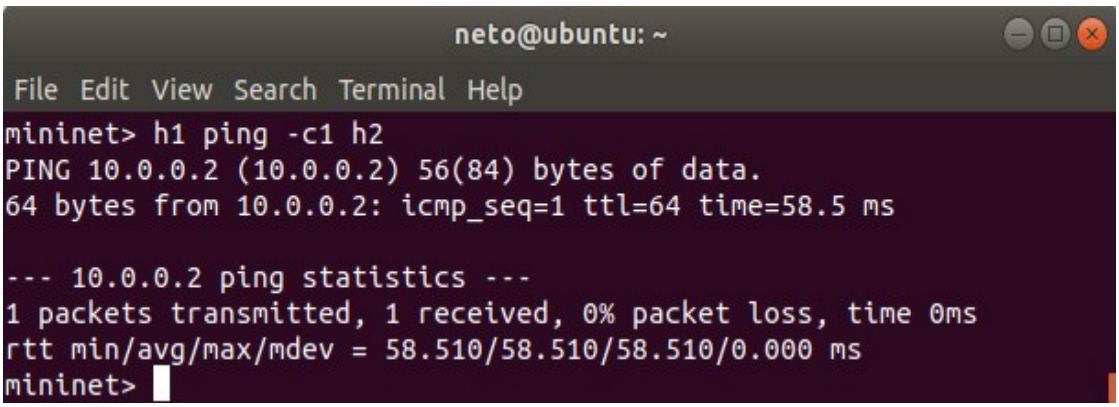

Figura 8. Comunicação entre os *hosts* h1 e h2 para que o *switch* receba pacotes.

4 Testes e Resultados

4.1 Execução do controlador e *switch OpenFlow* (Superclasse *example_switch_13*)

Após a criação do ambiente a ser trabalhado, é possível analizar tudo o que acontece no *switch*.

```
neto@ubuntu:~$ sudo mn --custom ~/mininet/custom/topo-1sw-4host.py --topo mytopo
 --mac --switch ovsk --controller remote
[sudo] password for neto:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Figura 9. Criando o ambiente.

Antes de executar o algoritmo, podemos observar o status do *switch*.

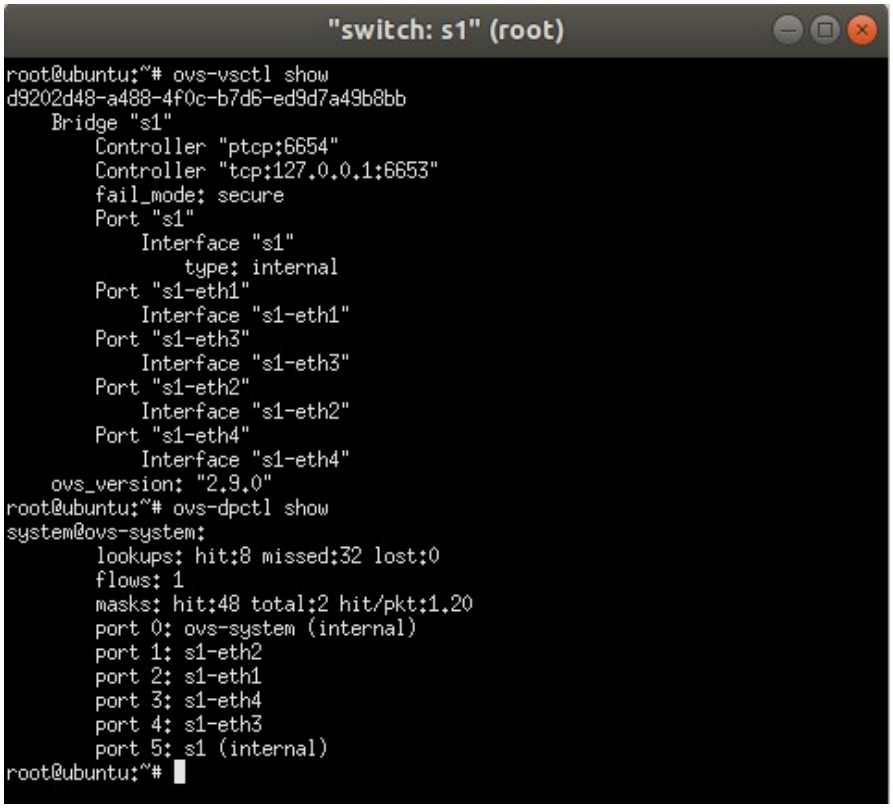

Figura 10. Status do *Open vSwitch*.

O *switch* (*bridge*) s1 foi criado e quatro portas correspondentes aos hosts foram adicionadas. Após isto, é necessário determinar a versão *OpenFlow* que será utilizada no *switch*.

Figura 11. Determinando a versão do protocolo *OpenFlow.*

Neste momento, pode-se observar que a tabela de fluxos do *switch* ainda está vazia.

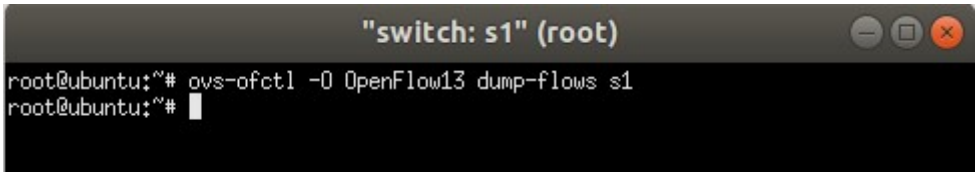

Figura 12. Tabela de fluxos do *switch* antes do início da transmissão de pacotes.

O próximo passo é executar o algoritmo do controlador.

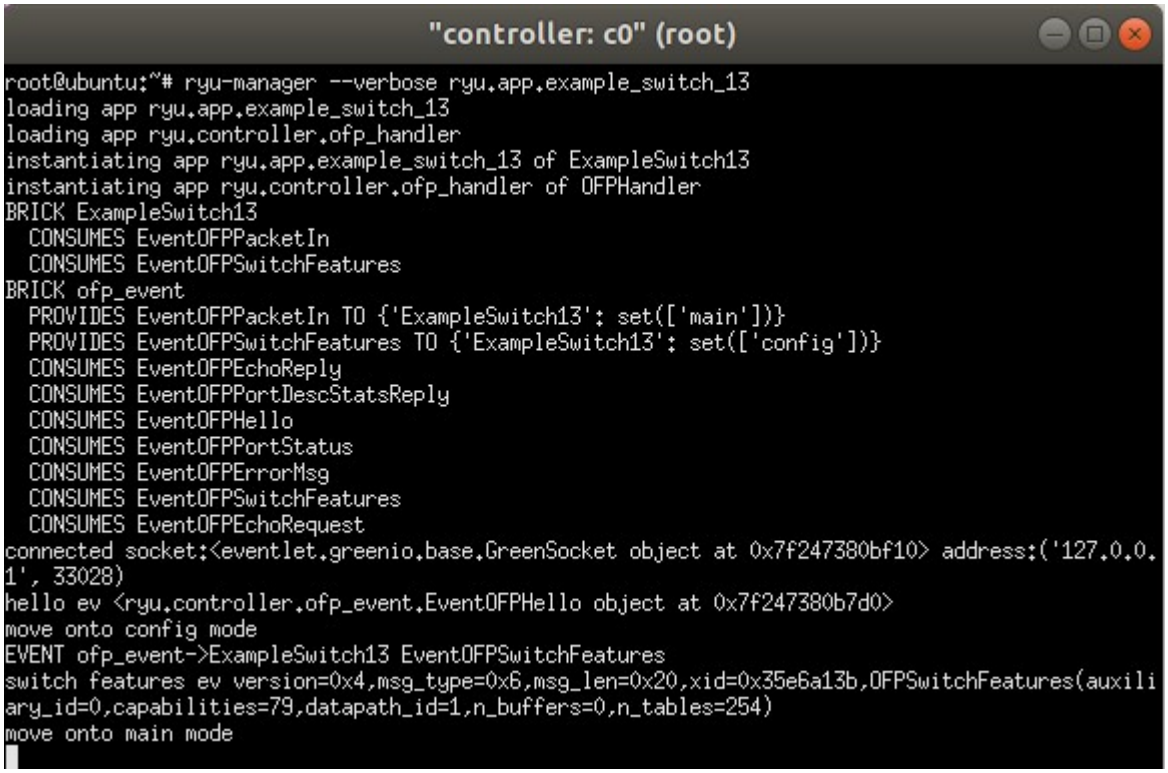

Figura 13. Iniciando o controlador.

Agora que o controlador foi iniciado e o *Open vSwitch* está conectado, a conexão entre eles foi estabelecida, um novo fluxo foi adicionado na tabela de fluxos do *switch* (enviar todos os pacotes para o controlador) e o *switch* encontra-se no status de aguardando pacote.

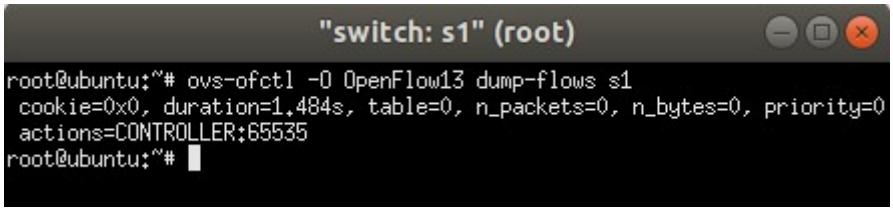

Figura 14. Tabela de fluxos do *switch* com novo fluxo.

Enfim, pode-se dar início a comunicação entre os *hosts* para que o *switch* comece a receber os pacotes.

```
mininet> h1 ping -c1 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=44.2 ms

--- 10.0.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 44.288/44.288/44.288/0.000 ms
mininet>
```

Figura 15. Comunicação entre *hosts*.

Após a comunicação, nota-se que novos fluxos foram adicionados na tabela de fluxos do *switch*:

1. Porta de entrada: 1, MAC destino:00:00:00:00:00:03, *host* 1 → Ação: enviar pacote para porta 3;

2. Porta de entrada: 3, MAC destino:00:00:00:00:00:01, *host* 3 → Ação: enviar pacote para porta 1;

```
"switch: s1" (root)
root@ubuntu:~# ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=83.226s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth3",dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=83.205s, table=0, n_packets=1, n_bytes=42, priority=1,in_port="s1-eth1",dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=1411.960s, table=0, n_packets=19, n_bytes=1302, priority=0 actions=CONTROLLER:65535
root@ubuntu:~#
```

Figura 16. Tabela de fluxos com novos fluxos.

Por parte do controlador, nota-se que durante a comunicação entre os *hosts*, foi realizado todas as tomadas de decisões descritas no capítulo anterior.

1. Pacote entrou pela porta 1 do *switch* e foi direcionado para o controlador;

2. Endereço MAC de destino ainda não reconhecido, o pacote é enviado para todas as portas do *switch*;

3. Após aprendido o endereço MAC destino, uma mensagem de reconhecimento é enviada pelo *host*;

4. Pacote é redirecionado para porta destino.

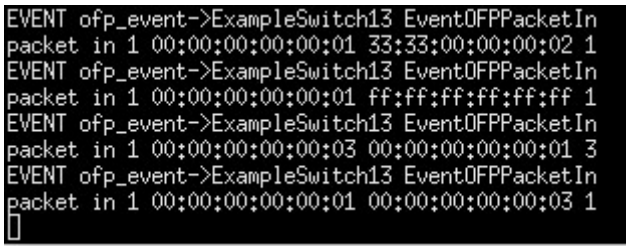

Figura 17. Tomada de decisão do controlador para o pacote recebido.

4.2 Análise e coleta de dados (Subclasse *simple_monitor_13)*

Sendo uma subclasse de *exemple_switch_13*, esta classe herda todos os métodos e variáveis de intância da superclasse. Com isso, ao executar esta classe, uma instância da superclasse será criada e executada.

Possuindo os três primeiros passos idênticos demonstrados no Capítulo 3 (criação do ambiente, configuração do *Open vSwitch* e inicialização do controlador para análise e coleta de dados), é iniciada a demonstração dos dados obtidos pelo controlador.

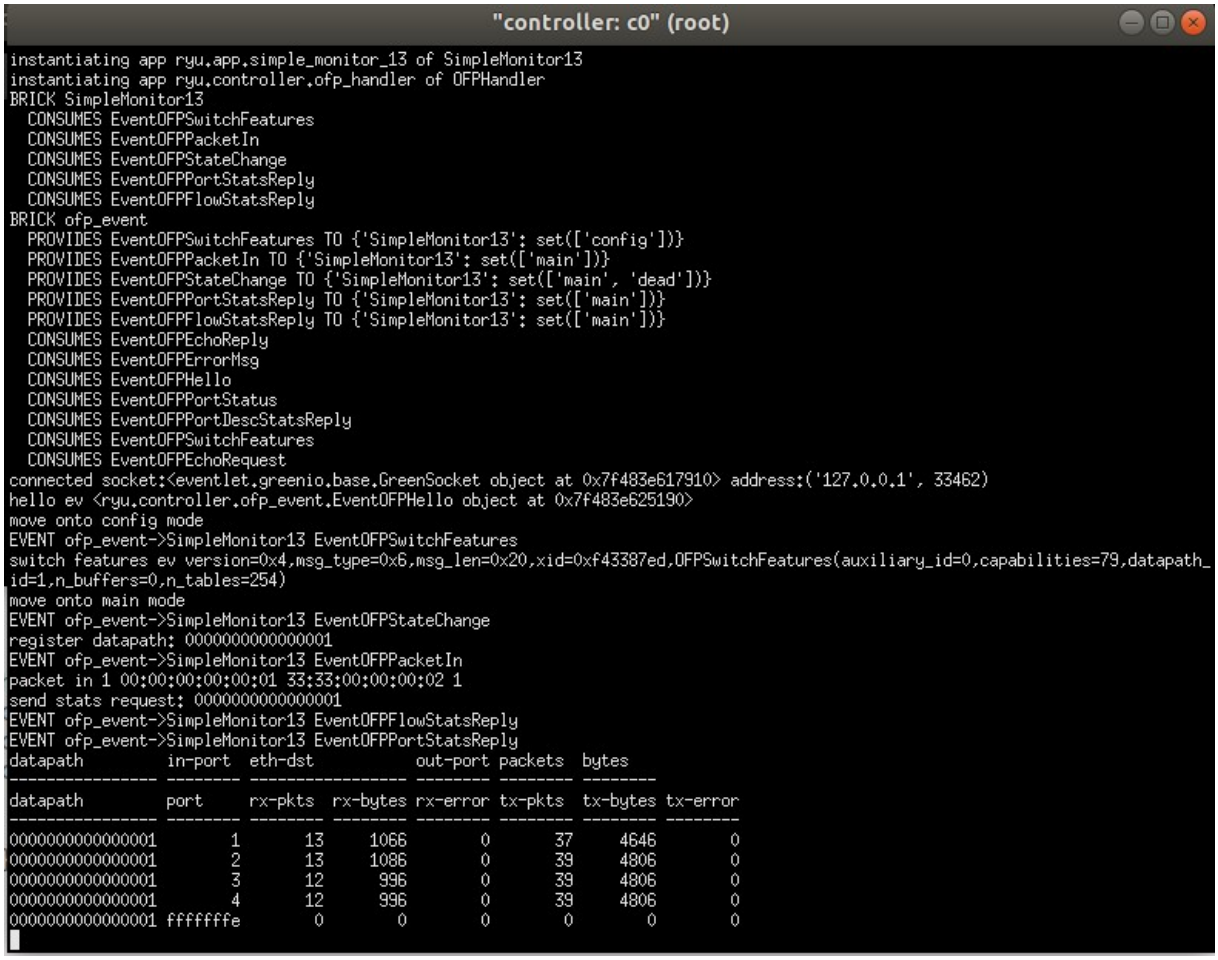

Figura 18. Dados coletados incialmente pelo controlador.

Neste ponto, pode-se observar que a contagem de pacotes já foi iniciada e que nenhum fluxo foi adicionado a tabela de fluxos..

Executando o comando de comunicação entre os *hosts* é possível observar a mundaça na numeração de *bytes* transferidos entres os *hosts* e a adição de um novo fluxo de endereçamento:

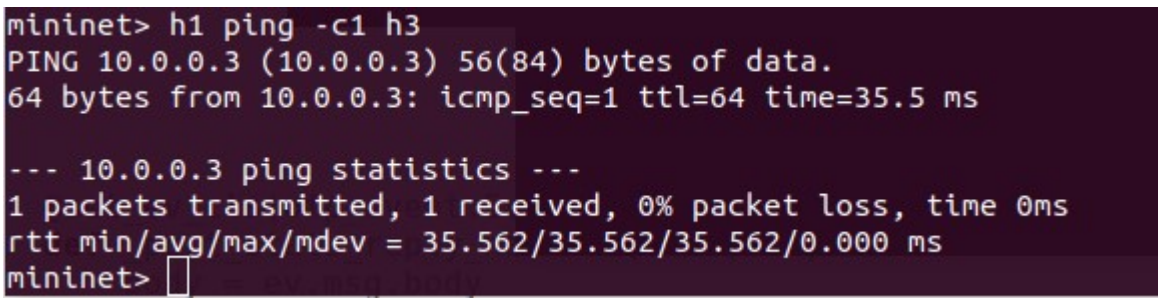

Figura 19. Comunicação entre *hosts*.

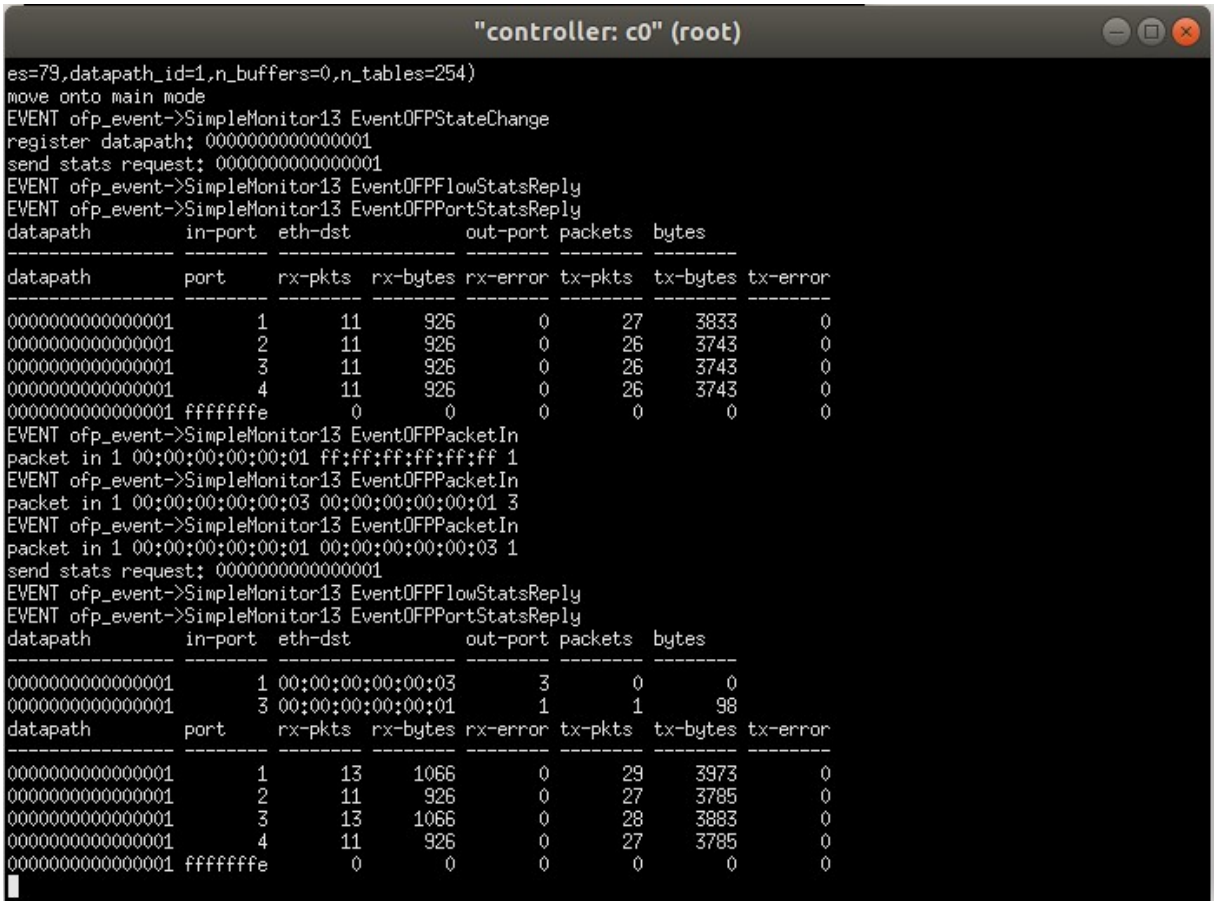

Figura 20. Dados coletados pelo controlador com nova tabela de fluxos.

De acordo com a tabela de fluxos, pode-se observar os novos fluxos e as novas informações estatísticas a respeito dos pacotes transferidos entre os *hosts* utilizados:

1. Porta de entrada: 1, MAC destino:00:00:00:00:00:03, porta de saída: 3;

2. Porta de entrada: 3, MAC destino:00:00:00:00:00:01, porta de saída: 1;

5 Conclusão e Trabalhos Futuros

5.1 Conclusão

Por conta do constante crescimento da população, as redes de computadores ao longo do anos vêm se desenvolvendo cada vez mais com novos meios de utilização, fazendo com que os servidores de internet sejam cada vez mais requisitados. Com a utilização de redes definidas por *software*, grandes possibilidades para o controle e manipulação da redes atuais podem ser desenvolvidos. Este projeto propõe um método de controle de redes de computadores que utiliza como base o protocolo *OpenFlow* para conseguir produzir uma rede definida por *software*, onde um controlador toma todas as decisões sobre o direcionamento de pacotes de internet para serem realizados por um *switch*, sem a necessidade de um auxílio externo.

Tendo em vista que foi demonstrado apenas uma possibilidade de utilização de redes definidas por *software* em conjunto com o protocolo *OpenFlow*, com a proposta deste trabalho é possível desenvolver melhorias expressivas para as redes de computadores comuns encontradas nos dias atuais, como por exemplo, a proposta citada no capítulo a seguir.

5.2 Trabalhos Futuros

Como trabalhos futuros desta proposta, baseado no algoritmo apresentado, pode ser desenvolvido um novo algoritmo com o intuito de alcançar o balanceamento de carga entre os servidores utilizados na rede. Por exemplo, em uma rede de computadores que possua mais de um servidor para o recebimento de pacotes por parte dos *hosts*, apenas um dos servidores é utilizado de inicio. Determina-se um limiar de carga a ser utilizado pelos servidores para que quando o limiar de carga do servidor que estiver sendo utilizado for ultrapassado, o controlador irá determinar as instruções necessárias para que os novos pacotes sejam redirecionados para um outro servidor que não esteja em uso. Com isso é possível

ocorrer uma distribuição uniforme de carga entre os *hosts* e os servidores, com o intuito de evitar o congestionamento da rede e a sobrecarga dos servidores utilizados.

Referências Bibliográficas

[1] Cui Chen-xiao, Xu Ya-bin. Research on Load Balance Method in SDN. Dissertação da escola de computação da Universidade de Ciências da Informação e Tecnologia de Beijing, 2016.

[2] Leandro Adrioli, Rodrigo da Rosa Righi, Mateus Rauback Aubin. Analisando métodos e oportunidades em redes definidas por software (SDN) para otimizações de tráfego de dados, 2017.

[3] Dorgival Guedes, Luiz Felipe Menezes Vieira, Marcos Menezes Vieira, Henrique Rodrigues, Rogério Vinhal Nunes. Redes Definidas por Software: uma abordagem sistêmica para o desenvolvimento das pesquisas em Redes de Computadores, 2014.

[4] Rabee Mustapha Abuteir, Anne Fladenmuller, Olivier Fourmaux. SDN Based Architecture to Improve Video Streaming in Home Networks, 2016.

[5] Chin-Feng Lai, Ren-Hung Hwang, Han-Chieh Chao, Mohammad Mehedi Hassan, Atif Alamri. A Buffer-Aware HTTP Live Streaming Approach for SDN-Enable 5G Wireless Networks, 2015.

[6] Wolfgang Braun, Michael Menth. Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices, 2014.

[7] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogério Salvador, Maurício Ferreira Magalhães. OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes, 2011.

[8] Suqiao Li. Network Traffic Control and Bandwidth Management in Internet: A differentated Services Case Study, 1999.

[9] Kevin Lai, Mary Baker. Measuring Bandwidth, 1999.

[10] Gabriel Henrique Montezelo. Práticas Laboratoriais em SDN usando Mininet e POX, 2017.

[11] Open Network Foundation. SDN Architecure, 2014.

[12] P. Beaulah Soundarabai, Sandhya Rani A., Ritesh Kumar Sahai, Thriveni J., K.R. Venugopal, L.M. Patnaik. Comparative Study on Load Balancing Techniques in Distributed Systems, 2012.

[13] Payal Beniwal, Atul Garg. A comparative study of static and dynamic Load Balancing Algorithms, 2014.

[14] César Angonese. Balanceamento de carga de trabalho em computação em nuvem baseado em redes magnéticas virtuais, 2012.

[15] Sandeep Sharma, Sarabjit Singh, Meenakshi Sharma. Performance analysis of load balancing algorithms, 2008.

[16] Jyoti Vadhita, Anant Kumar Jayswal. Comparative study of load balancing algorithms, 2013.

[17] Bruno Silva. Orientação a Objetos, 2015.

[18] Grupo PET-Tele. Tutorial de Programação Orientada a Objetos, 2009.

[19] Prof. Carlos Alberto Kamienski. Introdução ao paradigma de orientação a objetos, 1996.

Apêndice A

Algoritmo computacional utilizado para a implementação de um *switch* inteligente com tabela de aprendizado sobre informações dos pacotes de rede (*ExampleSwitch13*):

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet


class ExampleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(ExampleSwitch13, self).__init__(*args, **kwargs)
        # initialize mac address table.
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # install the table-miss flow entry.
        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # construct flow_mod message and send it.
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
        datapath.send_msg(mod)

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # get Datapath ID to identify OpenFlow switches.
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        # analyse the received packets using the packet library.
        pkt = packet.Packet(msg.data)
        eth_pkt = pkt.get_protocol(ethernet.ethernet)
        dst = eth_pkt.dst
        src = eth_pkt.src

        # get the received port number from packet_in message.
        in_port = msg.match['in_port']

        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        # if the destination mac address is already learned,
        # decide which port to output the packet, otherwise FLOOD.
```

```
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        # construct action list.
        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time.
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
            self.add_flow(datapath, 1, match, actions)

        # construct packet_out message and send it.
        out = parser.OFPPacketOut(datapath=datapath,
                                  buffer_id=ofproto.OFP_NO_BUFFER,
                                  in_port=in_port, actions=actions,
                                  data=msg.data)
        datapath.send_msg(out)
```

Apêndice B

Algoritmo computacional utilizado para o monitoramento e coleta de dados estatísticos da rede (*simple_monitor_13*).

```
from operator import attrgetter

from ryu.app import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub


class SimpleMonitor13(simple_switch_13.SimpleSwitch13):

    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)
        self.datapaths = {}
        self.monitor_thread = hub.spawn(self._monitor)

    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        datapath = ev.datapath
        if ev.state == MAIN_DISPATCHER:
            if datapath.id not in self.datapaths:
                self.logger.debug('register datapath: %016x', datapath.id)
                self.datapaths[datapath.id] = datapath
```

```
        elif ev.state == DEAD_DISPATCHER:
            if datapath.id in self.datapaths:
                self.logger.debug('unregister datapath: %016x', datapath.id)
                del self.datapaths[datapath.id]

    def _monitor(self):
        while True:
            for dp in self.datapaths.values():
                self._request_stats(dp)
            hub.sleep(10)

    def _request_stats(self, datapath):
        self.logger.debug('send stats request: %016x', datapath.id)
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        req = parser.OFPFlowStatsRequest(datapath)
        datapath.send_msg(req)

        req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
        datapath.send_msg(req)

    @set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
    def _flow_stats_reply_handler(self, ev):
        body = ev.msg.body

        self.logger.info('datapath         '
                         'in-port  eth-dst           '
                         'out-port packets  bytes')
        self.logger.info('---------------- '
                         '-------- ----------------- '
                         '-------- -------- --------')
        for stat in sorted([flow for flow in body if flow.priority == 1],
                           key=lambda flow: (flow.match['in_port'],
                                             flow.match['eth_dst'])):
            self.logger.info('%016x %8x %17s %8x %8d %8d',
                             ev.msg.datapath.id,
                             stat.match['in_port'], stat.match['eth_dst'],
                             stat.instructions[0].actions[0].port,
                             stat.packet_count, stat.byte_count)

    @set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
    def _port_stats_reply_handler(self, ev):
        body = ev.msg.body

        self.logger.info('datapath         port     '
                         'rx-pkts  rx-bytes rx-error '
                         'tx-pkts  tx-bytes tx-error')
        self.logger.info('---------------- -------- '
                         '-------- -------- -------- '
                         '-------- -------- --------')
        for stat in sorted(body, key=attrgetter('port_no')):
            self.logger.info('%016x %8x %8d %8d %8d %8d %8d %8d',
                             ev.msg.datapath.id, stat.port_no,
                             stat.rx_packets, stat.rx_bytes, stat.rx_errors,
                             stat.tx_packets, stat.tx_bytes, stat.tx_errors)
```