

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/324780227>

Aplicação de Redes Definidas por Software em Internet das Coisas

Thesis · April 2018

DOI: 10.13140/RG.2.2.34729.49765

CITATIONS

0

READS

145

1 author:



Joab Silva

Instituto Federal de Educação, Ciência e Tecnologia da Paraíba

4 PUBLICATIONS 0 CITATIONS

[SEE PROFILE](#)

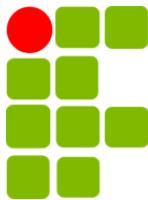
Some of the authors of this publication are also working on these related projects:



Network [View project](#)



Telecommunications [View project](#)



Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Coordenação do Curso Superior de Tecnologia em Telemática

APLICAÇÃO DE REDES DEFINIDAS POR *SOFTWARE* EM INTERNET DAS COISAS

Joab de Araújo Silva

Orientador: Marcelo Portela Sousa

Campina Grande, Abril de 2018

⑧ Joab de Araújo Silva



Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Campus Campina Grande
Coordenação do Cursos Superior de Tecnologia em Telemática

APLICAÇÃO DE REDES DEFINIDAS POR *SOFTWARE* EM INTERNET DAS COISAS

Joab de Araújo Silva

Monografia final apresentada à Coordenação
do Curso de Telemática do IFPB - Campus
Campina Grande, como requisito para con-
clusão do curso de Tecnologia em Telemática.

Orientador: Marcelo Portela Sousa

Campina Grande, Abril de 2018

S586a Silva, Joab de Araújo.
Aplicação de redes definidas por *software* em internet das coisas. - Campina Grande, 2018.
46f. : il.

Monografia (Graduação em Tecnologia em Telemática)
- Instituto Federal da Paraíba, 2018.
Orientadora: Prof. Marcelo Portela Sousa.

1. Redes Definidas por Software . 2. Internet das coisas.
3. *Open vSwitch* . I. Título.

CDU 004.41

APLICAÇÃO DE REDES DEFINIDAS POR *SOFTWARE* EM INTERNET DAS COISAS

Joab de Araújo Silva

DSc. Marcelo Portela Sousa
Orientador

DSc. Katyusco de Farias Santos
Membro da Banca

MSc. Moacy Pereira da Silva
Membro da Banca

Campina Grande, Paraíba, Brasil
Abril/2018

Dedico a Deus por ter me sustentado todo esse tempo,
a Ele toda honra e toda glória
e também a todos que de forma direta ou indiretamente
são responsáveis por eu chegar até aqui.

*Em um estado sombrio nós nos encontramos...
um pouco mais de conhecimento
iluminar nosso caminho pode.*

Mestre Yoda

*Faça ou não faça.
A tentativa não existe.*

Mestre Yoda

No sacrifice, no victory

Hammerfall

Agradecimentos

Agradeço primeiramente a Deus por toda força, sabedoria, paciência, pelas pessoas e professores que Ele em sua sabedoria, colocou no meu caminho eu chegar até onde cheguei, pois tenho certeza que sem essas coisas eu não teria chegado até aqui.

Agradeço também aos meus pais e família, pois desde minha infância sempre me incentivaram a estudar, essa vitória também é deles.

Agradeço ao meu grupo de amigos denominado "O Quinteto", Marcela Tassyany, Yngrid Keila, Edlane Oliveira e Whasley Cardoso, por toda cumplicidade, amizade e companheirismo desde o início do curso até agora no fim, em que nos formaremos todos juntos, são amizades que levarei para o resto da minha vida. Amo vocês!

Agradeço ao meu orientador Marcelo Portela, que é meu mentor, conselheiro, amigo, pai e exemplo de profissional que tenho e que tenha certeza que todas as minhas conquistas que alcancei e irei alcançar, tem muito mérito dele nisso.

Agradecimento a Dona Sônia por ter me puxado a orelha para eu voltar a estudar, e com isso chegar onde cheguei aqui. Também agradeço aos filhos dela Daniel, Dinho e Denilson por me acolherem como irmão e ao Seu Lula, por me acolher como filho, graças ao apoio de vocês cheguei aqui.

Agradeço a minha irmã do coração Renaly Oliveira, por me aguentar nos momentos difíceis e por estar sempre presente quando precisei, e por todo incentivo e por sempre aumentar meu astral quando foi necessário. Te amo Naly!

Agradeço a cada professor e funcionário do IFPB-CG que de forma direta e indireta contribuíram para eu chegar onde cheguei!

Agradeço ao pessoal do Ramo Estudantil IEEE do IFPB-CG, devo muito a essa instituição por meu amadurecimento acadêmico e como profissional, a experiência de voluntariado é de grande valor para a vida de um estudante.

Agradeço muito as amizades que fiz no decorrer do meu tempo no IFPB, em especial a Bianca Valentim(Filha), Gabrielle Brito(Veveta), Vitória Silva(Vivi) e Larissa Melo(Lari The Scientist), amigas que ganhei no IFPB para a vida e que mesmo depois de terem saído, estamos mais próximos a cada dia que se passa, vocês são as garotas da minha vida, amo vocês!

Agradeço a Emilly Batista e Alice Fortunato, pelos conselhos, consolos e puxadas de orelha quando foi necessário, e também por sempre verem o melhor que há em mim, mesmo quando eu não conseguia ver. Amo vocês!

A Letícia Guerra(Lê/Branca) por sempre ver o melhor que há em mim, e sempre fazer questão que eu notasse isso. Por sempre tentar levantar meu astral, por levantar minha moral, por levantar minha alta estima, por sempre salvar o meu dia com o seu sorriso. Obrigado Lê por ser maravilhosa comigo, mesmo a gente passando por uns momentos complicados, o laço e o amor que há entre a gente nunca se quebrou e não irá quebrar. Te amo!

Agradeço também a minha psicóloga pessoal Daniella Oliveira, por sempre está disposta a batermos aquele velho e bom papo que temos a mais de 10 anos de amizade. Agradeço pelos conselhos e por sempre ouvir meus problemas no decorrer desses mais de uma década de amizade!

Agradeço também as minhas amigas de loucuras, de filmes e séries Larissa Santos e Ellyda Soares, duas pessoas que temos muitas histórias para contar, duas pessoas que já compartilhamos vários momentos de nossas vidas, duas pessoas que sei muito delas e elas muito de mim, que me conhecem muito e que eu as conheço muito. Amo vocês! Minha vida sem vocês é chata!

Agradecimento ao melhor abraço que sinto tanta falta, a Clara Delfino, que tá lá na Suécia, mas minha admiração e amor que tenho por ela não diminuíram. Sinto muita falta sua, e saiba que aquele seu sorriso diário quando vinha em minha direção e o seu abraço amigo de todos os dias, faz uma grande falta nos meus dias. Te amo Clarinha!

E agradeço à todos que de alguma forma me ajudaram a chegar aqui!

Resumo

O sucesso da comunidade de tecnologia de redes pode ser considerado estrondoso, isso é notado pois permeia todos os níveis da sociedade, mas com isso trouxe problemas para a comunidade de pesquisadores, pois a inserção de protocolos e tecnologias novas tem se tornado inviáveis por causa dos riscos de interrupções na Internet. Surgiu assim as Redes Definidas por Software(*Software Defined Networking* - SDN) com a proposta para desenvolvimento de novas arquiteturas de Internet, baseada na separação do plano de dados e do controle, deixando assim toda lógica de controle da rede em um controlador, e em paralelo, houve o crescimento da Internet das Coisas(*Internet of Things* - IoT), que possibilita a integração de várias tecnologias em reunir, processar, decidir e transmitir dados. Neste trabalho serão apresentados os conceitos de SDN, IoT, além de apresentar o controlador ONOS e o protocolo *OpenFlow* que será configurado por meio do *Open vSwitch*. O objetivo desse trabalho é trabalhar com Raspberries Pi funcionando como Switches *OpenFlows* e um deles como trabalhando com uma aplicação IoT. Ao final serão apresentados os resultados gerados, os quais serão discutidos.

Palavras-chave: SDN, OpenFlow, ONOS, Open vSwitch, IoT, Raspberry Pi.

Abstract

The success of the network technology community can be considered as resounding, this is noticed because it permeates all levels of society, but with this has brought problems for the research community, as the insertion of new protocols and technologies has become impracticable because of the risks of Internet disruption. Thus emerged in Software Defined Networking (SDN) with the proposal for the development of new Internet architectures, based on the separation of the data plane and the control, thus leaving all control logic of the network in a controller, and in parallel, there was the growth of the Internet of Things (IoT), which enables the integration of various technologies in gathering, processing, deciding and transmitting data. In this work the concepts of SDN, IoT will be presented, in addition to presenting the controller ONOS and the protocol OpenFlow that will be configured through the Open vSwitch. The purpose of this work is to work with Raspberries Pi running as OpenFlows Switches and one of them as a working node with an IoT application, and ONOS providing network resources to that node. At the end will be presented the results generated, which will be discussed.

Keywords: SDN, OpenFlow, ONOS, Open vSwitch, IoT, Raspberry Pi.

Sumário

Lista de Abreviaturas	xiii
Lista de Figuras	xv
Lista de Tabelas	xvii
1 Introdução	1
1.1 Justificativa e Relevância do Trabalho	1
1.2 Objetivos	2
1.2.1 Objetivo Geral	2
1.2.2 Objetivos Específicos	2
1.3 Metodologia	2
1.4 Organização do Documento	3
2 Fundamentação Teórica	5
2.1 <i>Software Defined Networking</i> - SDN	5
2.2 Mininet	6
2.3 <i>Open Network Operating System</i> - ONOS	7
2.4 <i>Internet of Things</i> - IoT	8
2.5 <i>OpenFlow</i>	10
2.6 Open vSwitch	11
3 Descrição dos Métodos Utilizados	13
3.1 Instalação, Interface e Comandos Básicos do ONOS	13
3.2 Implantação do Switch <i>OpenFlow</i> usando o Open vSwitch	14
3.3 Rede Montada	15
3.3.1 Rede com VMs	15
3.3.2 Rede com Raspberry Pi	17
3.4 Mensagens Esperadas	20
4 Resultados	24
5 Discussão dos Resultados	30

6 Considerações Finais	31
7 Sugestões para trabalhos futuros	32
A Instalação ONOS	33
A.1 Pré-requisitos	33
A.2 Instalação	34
B Implementação dos Switches <i>OpenFlows</i>	36
B.1 Instalação do <i>Open vSwitch</i>	36
B.2 Configuração do <i>Open vSwitch</i>	36
C Configuração DHCP ONOS	37
C.1 Instalação e configuração do servidor DHCP	37
C.2 Ativação e configuração do ONOS para controle do serviço DHCP	37
D Configuração do Nô IoT	39
D.1 Esquema de Pinagem	39
D.2 Material usado	39
D.3 Código Embarcado da aplicação do Nô IoT	40
Referências Bibliográficas	45

Lista de Abreviaturas

SDN	<i>Software Defined Networking</i>
IoT	<i>Internet of Things</i>
ONOS	<i>Open Network Operating System</i>
QoS	<i>Quality of Service</i>
REST	<i>Representational State Transfer</i>
ONF	<i>Open Networking Foundation</i>
API	<i>Application Programming Interface</i>
HA	<i>High Availability</i>
OVSDB	<i>Open vSwitch Database</i>
Netconf	<i>Network configuration</i>
IPv6	<i>Internet Protocol version 6</i>
M2M	<i>Machine-to-Machine</i>
OvS	Open vSwitch
VM	<i>Virtual Machine</i>
L2	<i>Layer 2</i>
L3	<i>Layer 3</i>
ACL	<i>Access Control Lists</i>
sFlow(R)	<i>sampled Flow</i>
IPFIX	<i>Internet Protocol Flow Information Export</i>
SPAN	<i>Switch Port Analyzer</i>
RSPAN	<i>Remote SPAN</i>
GRE	<i>Generic Routing Encapsulation</i>
VXLAN	<i>Virtual Extensible LAN</i>
STT	<i>Stateless Transport Tunneling</i>
Geneve	<i>Generic Network Virtualization Encapsulation</i>
IPsec	<i>Internet Protocol Security</i>
SO	Sistema Operacional
UUID	<i>Universal Unique Identifier</i>
VIFs	<i>Virtual Interfaces</i>
PIFs	<i>Physical Interfaces</i>
DHCP	<i>Dynamic Host Configuration Protocol</i>

SYN	<i>Synchronize</i>
ACK	<i>Acknowledgement</i>
TCP	<i>Transmission Control Protocol</i>
SCTP	<i>Stream Control Transmission Protocol</i>
TLS	<i>Transport Layer Security</i>
OFPT	<i>OpenFlow based Parallel Transport</i>
MAC	<i>Media Access Control</i>
CLI	<i>Command Line Interface</i>
NAK	<i>No AcKnowledge</i>
USB	<i>Universal Serial Bus</i>
GUI	<i>Graphical User Interface</i>
SSH	<i>Secure Shell</i>
LED	<i>Light Emitting Diode</i>
LDR	<i>Light Dependent Resistor</i>

Listas de Figuras

2.1	Arquitetura SDN. Adaptado de [Kreutz <i>et al.</i> 2015]	6
2.2	Arquitetura ONOS. Adaptado de [Subramanian e Voruganti 2016]	7
2.3	Arquitetura IoT. Adaptado de [Hakiri <i>et al.</i> 2015]	9
2.4	Arquitetura <i>OpenFlow</i> . Adaptado de [Nunes <i>et al.</i> 2014]	10
2.5	Arquitetura Open vSwitch. Retirado de [Pfaff <i>et al.</i> 2009]	11
3.1	Rede Emulada pelo Mininet	14
3.2	Cenário com o Switch <i>OpenFlow</i> criado	16
3.3	Raspberry Pi 3 Model 3	17
3.4	Disposição dos Raqspberrys Pi	19
3.5	Ambiente montado	19
3.6	Three-Way Handshake	20
3.7	Mensagens OFPT_HELLO. Adaptado de [Flowgrammable 2012]	21
3.8	Mensagens OFPT_FEATURES_REQUEST e REPLY . Adaptado de [Flowgrammable 2012]	21
3.9	Mensagem OFPT_MULTIPART_REQUEST. Adaptado de [Flowgrammable 2012]	21
3.10	Pacote OFPT_MULTIPART_REQUEST. Fonte [Flowgrammable 2012] . . .	22
3.11	Mensagem OFPT_MULTIPART_REPLY. Adaptado de [Flowgrammable 2012]	22
3.12	Pacote OFPT_MULTIPART_REPLY. Fonte [Flowgrammable 2012] . . .	22
3.13	Processo DHCP. Retirado de [CCNA 2013]	23
4.1	Captura do <i>Handshake</i> TCP e a primeira mensagem <i>OpenFlow</i>	25
4.2	Requisição dos recursos do Switch <i>OpenFlow</i> e de resposta do Switch	26
4.3	Capacidades do Switch <i>OpenFlow</i>	26
4.4	Mensagens OFPT_MULTIPART_REQUEST e REPLY	27
4.5	Captura das mensagens DHCP	28
4.6	Captura da mensagem DHCP ACK no switch 1(s1)	29
A.1	CLI do ONOS	34
A.2	Interface WEB do ONOS	35
D.1	Pinos do Raspberry Pi 3 Model B e suas funções. Retirado de [Minatel 2017]	39

D.2 Esquema de pinagem e circuito usado nesse trabalho para o N�o IoT	40
---------------------------------------------------------------------------------	----

Lista de Tabelas

1.1	Cronograma de Atividades	3
3.1	Distribuição IP dos Raspberrys Pi	18
4.1	Descrição das capacidades do Switch <i>OpenFlow</i> 1.3	25

Capítulo 1

Introdução

O sucesso da comunidade de tecnologia de redes pode ser considerado estrondoso, pois permeia todos os níveis da sociedade, já que a maioria das atividades da sociedade usufrui dela. A grande demanda de uso da Internet trouxe problemas para a comunidade de pesquisadores, pois a inserção de novas tecnologias e pesquisas com novos protocolos, tornaram-se inviáveis devido ao risco de interrupções na Internet [Guedes *et al.* 2012] e consequentemente a interrupção de serviços que necessitem dela. A busca de melhorias e da estabilidade da Internet, são alvos principais da maior parte das pesquisas. Nesse contexto desafiador surgiu as Redes Definidas por Software(*Software Defined Networking* - SDN).

As Redes Definidas por Software, surgiram como proposta para desenvolvimento de novas arquiteturas de Internet, baseada na separação do plano de dados e do controle de interface de forma genérica e não proprietário [Liberato *et al.* 2014]. Em meio a essas arquiteturas de Internet, surgiu o conceito de Internet das Coisas (*Internet of Things* – IoT), que possibilita que várias tecnologias diferentes como sistemas embarcados, redes de sensores sem fio, computação na nuvem(*Cloud Computing*), possam reunir, processar, decidir e transmitir dados [Hakiri *et al.* 2015]. Como IoT possibilita que várias tecnologias trabalhem em conjunto e redes SDNs permite interoperabilidade entre fornecedores, pesquisas que implementem SDN em uma arquitetura de rede IoT, vem sendo desenvolvidas pela comunidade.

Este trabalho tem como objetivo implementar uma rede SDN com um nó IoT, e como metodologia de análise, será feito capturas dos pacotes da rede e mensagens *OpenFlow*, por meio do programa de capturas de pacotes *Wireshark*. A rede contará com Switches *OpenFlows* implementados em Raspberrys Pi e o ONOS(*Open Network Operating System*) para o controle lógico da rede, além de um nó IoT.

1.1 Justificativa e Relevância do Trabalho

Em uma época em que o acesso a Internet se tornou popular a demanda por serviços aumentou e com isso a necessidade de alta disponibilidade da rede tornou-se critério indispensável. Pesquisas precisam ser feitas e novos protocolos precisam ser desenvolvidos,

porém inserir novos protocolos ou mesmo até necessidade de alteração de hardware de alguns dispositivos que estão na rede, podem acarretar na interrupção de serviço.

SDN surgiu como uma solução prática para que tais interrupções possam não ocorrer, visto que pode-se fazer pesquisas e desenvolver protocolos em ambientes seguros e sem causar interrupções a grande rede da Internet.

Este trabalho implementará uma rede SDN em uma arquitetura IoT, para aplicação em um ambiente real, analisando os pacotes da rede e dos protocolos, usando quatro Raspberries Pi atuando como Switch *OpenFlow* e outro como nó IoT.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é aplicar uma rede SDN com um nó executando funções IoT, e monitorar o tráfego da rede, analisando as mensagens *OpenFlow* e também os protocolos da rede, em uma rede com quatro Switches *OpenFlows* implementados em Raspberries.

1.2.2 Objetivos Específicos

- Descrever as funcionalidades, modo de instalação e configuração do controlador ONOS;
- Descrever a configuração do Raspberry Pi no ambiente proposto;
- Projetar a rede SDN;
- Analisar os pacotes da rede e mensagens *OpenFlows* capturados pelo Wireshark.

1.3 Metodologia

A metodologia desse trabalho foi dividida em 5 etapas, que são:

- Etapa 1:
 - Revisão bibliográfica para embasamento teórico do trabalho e soluções correlatas existentes.
- Etapa 2:
 - Instalação do ONOS em uma máquina do Laboratório de Redes;
 - Integrar o ONOS com o emulador de redes Mininet;
 - Criar máquinas virtuais, para testes antes do uso dos Raspberry Pi;
- Etapa 3: Obter os primeiros resultados das capturas de pacotes para a defesa parcial;

- Etapa 4: Implementação das melhorias necessárias;
 - Implantar a rede com os Raspberry Pi.
- Etapa 5: Escrita e Defesa do TCC.

Na Tabela 1.1 é apresentado o cronograma com as etapas.

Etapa	Out	Nov	Dez	Jan	Fev	Mar
Revisão Bibliográfica	X					
Instalação do ONOS	X	X	X			
Resultado Parciais		X	X			
Implantação das melhorias			X	X	X	
Escrita e Defesa TCC						X

Tabela 1.1: Cronograma de Atividades

1.4 Organização do Documento

Na Seção 2.1, são apresentados os desafios que os administradores de rede enfrentam em ambientes de redes físicas e também os desafios quando se trata de novas pesquisas de protocolos de redes. Após isso é apresentado o conceito de SDN e suas aplicações, e como ele surgiu para solucionar tais problemas.

Na Seção 2.2, são apresentados os conceitos relacionados ao Mininet e suas características, além de suas aplicações em ambientes físicos e virtuais.

Na Seção 2.3, uma descrição sobre o ONOS é realizada, detalhando toda sua arquitetura e funcionamento. É apresentado também sua aplicação em ambiente real/virtual e em quais tipos de dispositivos ele pode ser instalado.

Na Seção 2.4, é apresentado a Internet das Coisas, mostrando o que é e algumas de suas características, além disso é apresentado como o SDN surgiu como um modo de simplificar a complexidade de redes com arquitetura IoT.

Na Seção 2.5, é definido o que é *OpenFlow* e sua utilidade em ambientes de pesquisa. É apresentado também a sua arquitetura e detalhado seu funcionamento.

Na Seção 2.6, o Open vSwitch é mostrado como solução alternativa ao baixo desempenho do *OpenFlow*, sua arquitetura é mostrada e explicada, além de ter seu funcionamento detalhado.

Na Seção 3.1, foi descrito como o ONOS foi utilizado neste trabalho, mostrando como inicializar e as ferramentas disponíveis.

Na Seção 3.2, é descrito como é a instalação do Open vSwitch e o passo a passo de como foi a criação do Switch *OpenFlow* neste trabalho.

Na Seção 3.3 são descritas as redes que foram montadas para a apresentação dos resultados.

Na Seção 3.4, é mostrado quais mensagens são esperadas na captura dos pacotes do tráfego de rede.

Na Seção 4, é apresentado o resultado das capturas feitas pelo *Wireshark* e a descrição dos mesmos.

Na Seção 5, serão discutidas os resultados apresentados na Seção 4

Na Seção 6, serão dadas as considerações finais.

Na Seção 7, serão apresentado os trabalhos futuros.

No Apêndice A, há o guia de instalação do ONOS.

No Apêndice B, contém um guia de como criar um Switch *OpenFlow*, usando o Open vSwitch.

No Apêndice C, é descrito os passos para o funcionamento do DHCP.

No Apêndice D, tem o código embarcado da aplicação IoT.

Capítulo 2

Fundamentação Teórica

2.1 *Software Defined Networking - SDN*

As redes de computadores normalmente são construídas com dispositivos, por exemplo como roteadores e switches, que possuem protocolos complexos. Os administradores de rede são responsáveis por configurar os protocolos de forma que possam atender aos requisitos da rede, assim adaptando-se às mudanças que podem ocorrer. Muitas vezes, eles implementam configurações complexas e com acesso a ferramentas muito limitadas, além do fato de que os dispositivos de rede são em sua maioria caixas pretas, assim aumentando o desafio que os administradores enfrentam [Nunes *et al.* 2014].

Devido a esses desafios, vários pesquisadores afirmam que arquiteturas de redes de computadores, atingiram um nível de amadurecimento que as tornaram pouco flexíveis [Guedes *et al.* 2012]. Na infraestrutura de rede encontra-se uma camada de software de controle, que é responsável pela pilha formada pelos protocolos [Rothenberg *et al.* 2010], porém tornou-se evidente a necessidade de melhoria da lógica de controle, mas qualquer mudança de configuração avançada do equipamento estão sujeitas a ciclos de desenvolvimento e de testes restritos ao fabricante do equipamento [Hamilton 2009]. Quando se trata de pesquisas, a exigência é maior, pois requer experimentos de novos protocolos e funcionalidades da rede em ambientes reais.

Nesse ambiente desafiador, o SDN, é um paradigma de rede que surge como uma forma de mudar as limitações das infraestruturas de rede, pois centraliza toda a lógica de controle da rede em um controlador, simplificando a implementação de políticas na rede [Kim e Feamster 2013], ou segundo a definição da ONF (*Open Networking Foundation*):

"Software-Defined Networking (SDN) é uma arquitetura emergente dinâmica, gerenciável, econômica e adaptável, tornando-a ideal para a alta largura de banda e dinâmica para as aplicações atuais. Essa arquitetura desacopla as funções de controle de rede e encaminhamento permitindo o controle de rede para se tornar diretamente programável, e a infraestrutura subjacente a ser abstraída para aplicações e serviços de rede. O protocolo OpenFlow é um elemento fundamental para a construção de soluções SDN " [ONF 2011].

A Figura 2.1 ilustra a arquitetura.

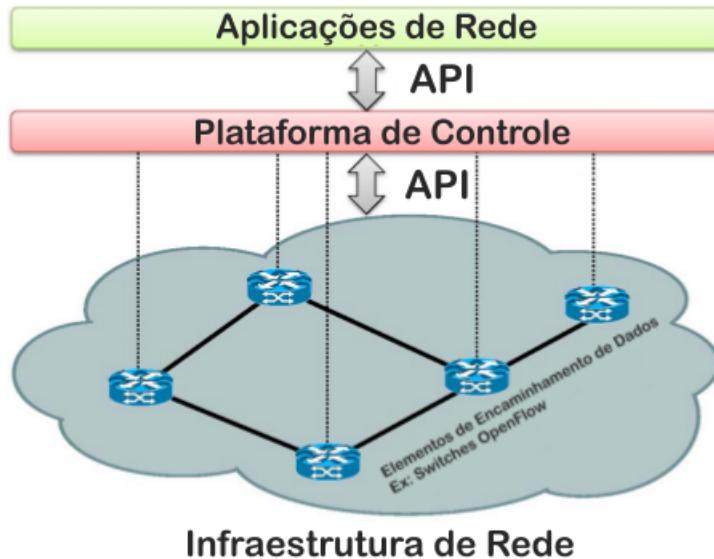


Figura 2.1: Arquitetura SDN. Adaptado de [Kreutz et al. 2015]

A separação do plano de controle(controlador SDN) e do plano de dados(Switches *OpenFlow*), é realizada por meio de uma interface de programação entre os switches e o controlador SDN [Kreutz et al. 2015]. Este controlador exerce controle direto sobre o plano de dados, por meio de uma interface de programação de aplicação (*Application Programming Interface - API*). O *OpenFlow* é um exemplo dessa API. Ele possui uma ou mais tabelas de regras de remoção de pacotes [McKeown et al. 2008], com cada regra correspondendo a um subconjunto do tráfego e executando ações como descartar, reencaminhar e modificar. Dependendo das regras implementadas pelo controlador, o *OpenFlow* pode ser instalado pelo controlador como um roteador ou executar outras funções como um平衡ador de cargas por exemplo.

Como já mencionado, é necessário a presença de um controlador e para isso existem *frameworks* que exercem esse papel. Nas seções seguinte serão apresentados o emulador de rede Mininet e o controlador ONOS.

2.2 Mininet

O Mininet é um emulador de rede responsável por criar uma rede de *hosts* virtuais. Os *hosts* Mininet executam um software de rede Linux e seus switches suportam o *OpenFlow* para roteamento. O Mininet é ideal para pesquisa, desenvolvimento, aprendizagem, protótipagem, testes, depuração e quaisquer tarefas que possam se beneficiar de ter uma rede experimental [Mininet 2018].

São características do Mininet:

- Fornecer teste de rede para o desenvolvimento de aplicações *OpenFlow*;

- Permitir que desenvolvedores concorrentes trabalhem sem interferir um com os outros na mesma topologia;
- Permitir testes de topologia, sem a necessidade de conexão com a rede física;
- Suportar topologias personalizadas e além de incluir um conjunto básico de topologias pré-feitas;
- Fornecer uma API Python para criação e experimentação de redes.

Ao desenvolver e testar uma rede personalizada no Mininet, para um controlador *OpenFlow*, essa rede pode ser implementada para um sistema real com mudanças mínimas, para ser testada em um ambiente real. Isso significa que um projeto que funciona no Mininet geralmente pode ser implantada em um ambiente real.

O Mininet pode trabalhar com controladores, o ONOS que é implementado neste trabalho, e descrito na seção 2.3.

2.3 *Open Network Operating System* - ONOS

O ONOS é o primeiro sistema operacional aberto de rede SDN dedicado especificamente a Provedor de Serviços. O ONOS é projetado para fornecer Alta Disponibilidade (High Availability - HA) [ONOS 2014]. Fornece o plano de controle para o SDN, gerenciando componentes de rede e executando programas de software ou módulos fornecendo serviços de comunicação para os *hosts* e outras redes. O ONOS e seus aplicativos atuam como um controlador SDN extensível, modular e distribuído [ONOS 2017].

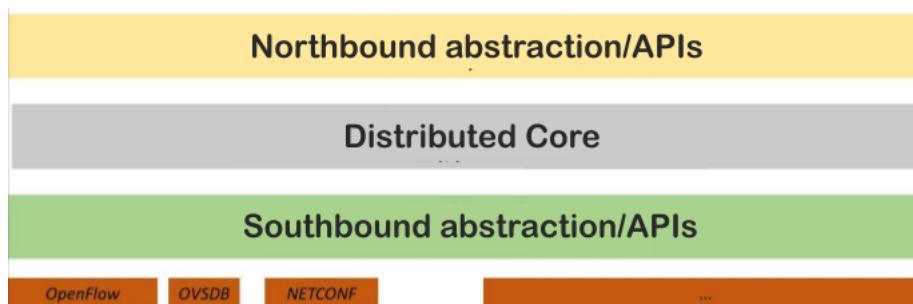


Figura 2.2: Arquitetura ONOS. Adaptado de [Subramanian e Voruganti 2016]

Na Figura 2.2, ilustra-se a arquitetura de funcionamento do ONOS, no qual os três principais recursos são detalhados a seguir [Subramanian e Voruganti 2016]:

- **Distributed Core:** O sistema operacional SDN foi projetado para ser executado em um *cluster* que implementa requisitos de rede como tolerância a falhas, alto desempenho, escalabilidade elástica com base em aplicativos e demandas de largura de banda.

- **Northbound abstraction/APIs:** Fornecem serviços de configuração e gerenciamento para o desenvolvimento de aplicativos SDN. Fornecem a representação gráfica da rede. Permitem que os aplicativos especifiquem seus requisitos de controle de rede sob a forma de política, facilitando o desenvolvimento de aplicativos. Eles também suportam abstrações de dispositivos.
- **Southbound abstraction/APIs:** Fornecem *plugins* de protocolo para se comunicarem com dispositivos de rede. O *OpenFlow*, OVSDB e Netconf são suportados por padrão. É uma arquitetura extensível e plugável, permitindo assim o suporte para que os protocolos se comuniquem com dispositivos legados, que isolam o Core dos diferentes protocolos de comunicação do dispositivo.

O ONOS pode ser implantado como um serviço em um *cluster* e o mesmo é executado em cada nó do *cluster*. O administrador de rede pode acrescentar servidores de forma incremental sem interrupção. Aplicativos e dispositivos da rede não precisam ter conhecimento se estão trabalhando com uma única instância ou com várias instâncias do ONOS. Com isso, esse recurso torna o ONOS escalável. É o núcleo distribuído que faz todo o trabalho de distribuição no *cluster*.

O núcleo distribuído é responsável por fornecer os serviços de mensagens, gerenciamento de estado e eleição de liderança entre as instâncias. Como resultado, as múltiplas instâncias comportam-se como uma única entidade, com isso permite que haja alto rendimento, baixa latência e disponibilidade do *cluster*. O estado operacional é gerenciado entre as instâncias usando vários mecanismos, que varia de acordo com o estado da instância, por exemplo, as tabelas de fluxo possuem tamanhos, padrões de leitura/gravação e durabilidade únicos.

O ONOS foi criado com os seguintes objetivos:

- Excluir a necessidade dos desenvolvedores de aplicativos de rede, de conhecer as complexidades do hardware proprietário;
- Permitir que os administradores de rede não precisem se preocupar com as complexidades operacionais de interfaces e protocolos proprietários.

O ONOS pode ser implantado em servidores, desktops e notebooks. O ONOS nesse trabalho será implantado em uma máquina virtual, em uma rede composta por um notebook e Raspberries Pi. Um Raspberry Pi irá atuar como nó IoT. Na seção 2.4 é apresentada a definição de IoT e sua arquitetura.

2.4 Internet of Things - IoT

Com o aumento de “coisas” na rede, IoT fez com que os padrões tradicionais de IP não pudesse se encaixar no grande número de dispositivos conectados. Essas coisas podem ter características e recursos diferentes, ou seja, há necessidade de mesclar protocolos de

roteamento para acomodar esse crescimento. O IPv6 apareceu como solução para lidar com esse número de dispositivos, mas não com a heterogeneidade das coisas [Jararweh *et al.* 2015]. O SDN já foi utilizado para permitir que diferentes coisas de distintas redes se comunicassem usando o IPv6 e, simplificando as operações de gerenciamento e controle, adicionando um controlador IoT sobre o controlador SDN [Martinez-Julia e Skarmeta 2014]. Depois de definir e estabelecer as regras de reencaminhamento, o controlador IoT encaminha essas regras para o controlador SDN que envia para os dispositivos de encaminhamento. Portanto, mesmo que essas coisas possuam protocolos incompatíveis, os dispositivos de encaminhamento traduzem de forma adequada os pacotes para que sejam entendidos pelo receptor. Isso permite que objetos heterogêneos na rede se comuniquem de forma eficiente.

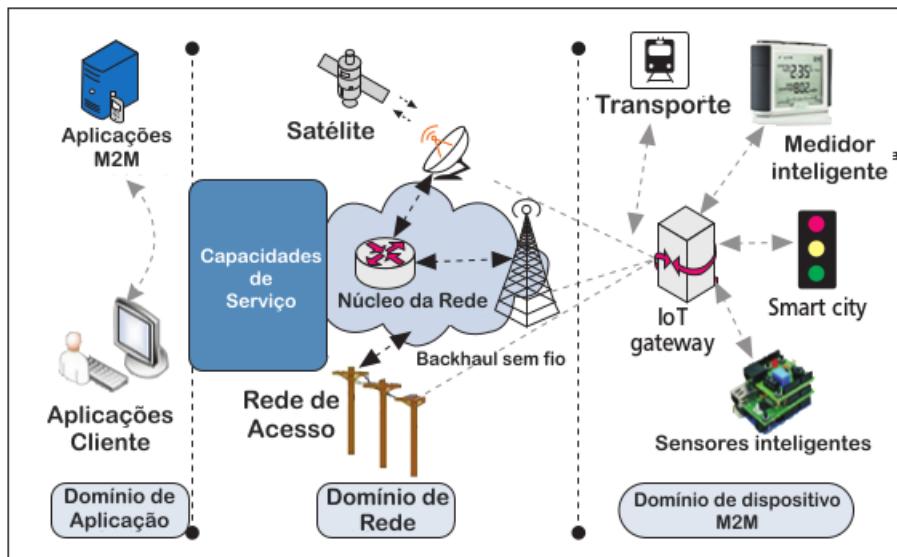


Figura 2.3: Arquitetura IoT. Adaptado de [Hakiri *et al.* 2015]

A Figura 2.3 ilustra uma arquitetura IoT de alto nível, composta por três domínios: do dispositivo, da rede e da aplicação [Hakiri *et al.* 2015]. No domínio do dispositivo, por meio da rede de acesso, o nó fornece conectividade ao domínio de rede.

O domínio de rede é formado por diferentes redes de acesso, fornecendo conectividade por meio de diversas tecnologias e também fornecendo conectividade à rede central, incluindo conectividade heterogênea e de multi-tecnologia.

Por fim, o domínio de aplicação que inclui as aplicações IoT e as infraestruturas servidor/nuvem, que compartilham seu conteúdo, possivelmente dando suporte à outros dispositivos, programas analíticos e/ou pessoas, também incluem recursos de serviço, que fornecem funções compartilhadas.

Neste trabalho, para montar uma rede SDN em IoT, será utilizado quatro Raspberries Pi que funcionarão como um Switches *OpenFlows*, neles serão instalado o *Open vSwitch* que implementa um switch virtual multi camadas, e com o protocolo *OpenFlow* habilitado que permitirá que um controlador remoto possa tomar as decisões [Pantuza 2017]. Na seção 2.5 será explicado o protocolo *OpenFlow*.

2.5 OpenFlow

O *OpenFlow* é um padrão aberto para SDN com o objetivo principal de permitir a utilização de equipamentos de redes comerciais para pesquisa e experimentação de novos protocolos de rede em paralelo com a operação normal das redes, ou seja, *OpenFlow* fornece um protocolo aberto para programar a tabela de fluxo em diferentes switches e roteadores, sendo assim um administrador de rede pode dividir o tráfego da rede em fluxos. Os pesquisadores podem controlar seus próprios fluxos, desta forma, podem testar novos protocolos de roteamento, por exemplo, na mesma rede, com o tráfego separado e sendo tratado da mesma maneira [Guedes *et al.* 2012] [McKeown *et al.* 2008].

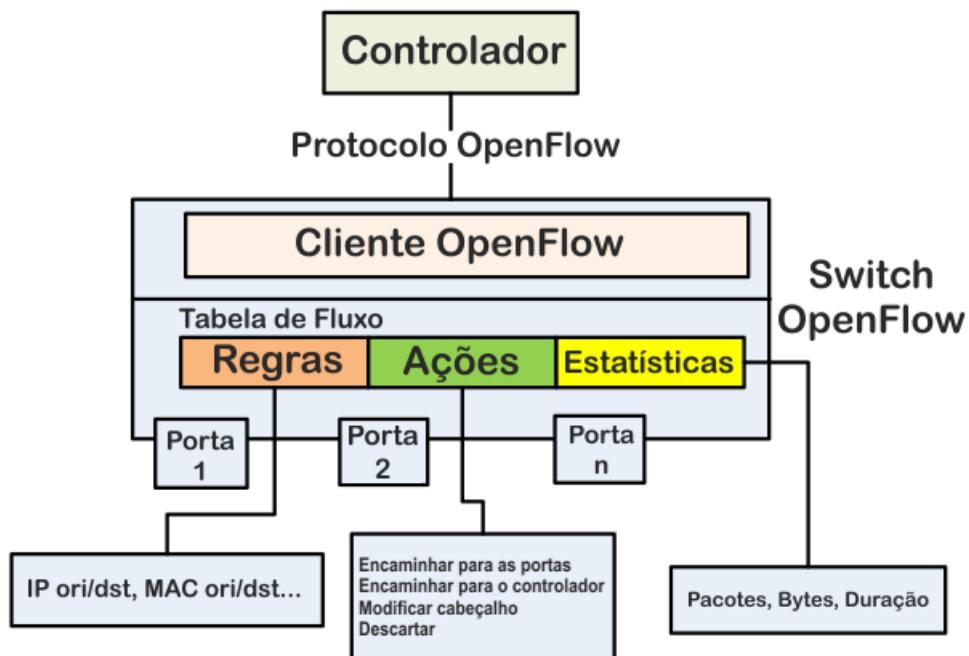


Figura 2.4: Arquitetura OpenFlow. Adaptado de [Nunes *et al.* 2014]

Na arquitetura *OpenFlow*, ilustrada na Figura 2.4, o Switch *OpenFlow* contém uma ou mais tabelas de fluxo(*flow tables*) e uma camada de abstração que se comunica com um controlador via protocolo *OpenFlow*. Os fluxos(*flows*) contidos nas tabelas de fluxo, determinam como os pacotes pertencentes a um fluxo serão processados e encaminhados [Nunes *et al.* 2014].

Normalmente as entradas de fluxo consistem em:

- **Campos de regras:** São usados para combinar pacotes recebidos, podem conter informações encontradas no cabeçalho do pacote, porta de entrada e metadados;
- **Estatísticas:** Usados para coletar estatísticas de um fluxo específico, como número de pacotes recebidos, número de *bytes* e duração do fluxo;
- **Conjunto de ações:** São aplicadas após um "match" e são eles que determinam como lidar com os pacotes correspondentes.

Quando um pacote chega em um Switch *OpenFlow*, os campos do cabeçalho do pacote são extraídos e combinados com os campos correspondentes das entradas da tabela de fluxo. Se uma entrada correspondente for encontrada, o switch aplicará o conjunto de regras correspondente.

Pode-se dizer que um Switch *OpenFlow* consiste em 3 partes [McKeown *et al.* 2008]: 1

- Uma Tabela de Fluxo, com uma ação associada a cada entrada de fluxo, para indicar ao switch como processar o fluxo; 2 - Um canal seguro que conecta o switch ao controlador; 3 - O protocolo *OpenFlow*, fornece uma maneira do controlador se comunicar com um switch.

O protocolo *OpenFlow* tem sido utilizado como base em diversos experimentos, mas necessita de um melhor desempenho [Guedes *et al.* 2012]. Nesse cenário o Open vSwitch, descrito na Seção 2.6, surgiu para suprir essa necessidade.

2.6 Open vSwitch

O Open vSwitch (OvS) é um switch virtual que segue a arquitetura *OpenFlow* [Guedes *et al.* 2012], e é um switch virtual multicamada, implantado em ambientes virtualizados que está incluído como recurso padrão do Linux 3.3 Kernel em diante [Subramanian e Voruganti 2016]. Open vSwitch reside no hypervisor ou domínio de gerenciamento (*Management Domain*), fornecendo conectividade entre as máquinas virtuais(*Virtual Machine* - VM) e as interfaces físicas [Pfaff *et al.* 2009].

Além de trazer recursos de aprendizagem L2(*Layer 2*), encaminhamento L3(*Layer 3*) e ACL(*Access Control Lists*). A OvS suporta qualidade de serviço (QoS), também suporta visibilidade na comunicação entre VM via NetFlow, sFlow(R), IPFIX, SPAN, RSPAN, e suporta protocolos de tunelamento como GRE, VXLAN, STT e Geneve, com IPsec [Pettit *et al.* 2010; Pfaff *et al.* 2009].

A arquitetura do Open vSwitch é apresentada na Figura 2.5, e tem como componentes principais:

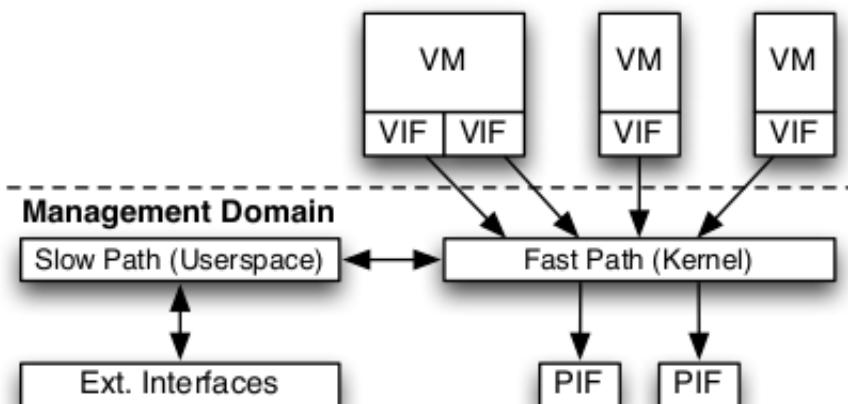


Figura 2.5: Arquitetura Open vSwitch. Retirado de [Pfaff *et al.* 2009]

- **Fast Path:** É um módulo no núcleo do SO (Sistema Operacional), que interage ativamente com o tráfego de rede, pois é responsável pelo encaminhamento dos pacotes de rede, pelas rotas encontradas na tabela de fluxos;
- **Slow Path:** É um componente do nível de usuário, é responsável por implementar as demais funcionalidades do plano de controle, como as interfaces de configuração do switch, a lógica de uma ponte com aprendizado (*learning bridge*) e as funcionalidades de gerência remota, etc.

Em uma implementação padrão, ele funciona como um switch L2, porém para suportar a integração em ambientes virtuais e permitir a distribuição de switch, o Open vSwitch exporta interfaces para manipular o estado de encaminhamento e gerenciar o estado de configuração. Tais interfaces são as de Configuração, Encaminhamento(*Forwarding Path*) e Gerenciamento de Conectividade [Pfaff *et al.* 2009].

Interface de Configuração: Por meio desta interface, um processo remoto pode ler e gravar o estado de configuração e configurar "triggers" para receber eventos assíncronos. Além disso, esta interface fornece comunicação entre as portas de rede e o ambiente virtual. Por exemplo, a interface expõe os identificadores globalmente únicos (*Universal Unique Identifier - UUID*) para VIFs (*Virtual Interfaces*) ligados ao switch.

Interface de Encaminhamento: O Open vSwitch fornece um método remoto para manipular o encaminhamento dos pacotes, permitindo assim que um processo externo seja gravado diretamente na tabela de encaminhamento, com isso pode-se decidir o encaminhamento de um pacote para uma ou mais portas, descartar o pacote ou encapsular/desencapsular o pacote. Essa interface implementa um superconjunto do protocolo *OpenFlow*.

Interface de Gerenciamento de Conectividade: Nesta interface a camada de virtualização pode manipular sua configuração topológica, isso inclui a criação de switches, gerenciando assim a conectividade VIF e PIF (*Physical Interfaces*). Em sua implementação mais simples, Open vSwitch é muito parecido com um switch físico dentro da camada de virtualização. Cada instância é administrada separadamente por meio dessas interfaces, assim garantindo visibilidade e controle na comunicação entre VMs. A capacidade de manipular a tabela de encaminhamento por meio de uma interface externa, permite que o estado de fluxo de baixo nível migre com uma máquina virtual, por exemplo, isso pode ser usado para migrar contadores de fluxo e ACLs existentes.

Capítulo 3

Descrição dos Métodos Utilizados

Nesse capítulo serão apresentados as ferramentas utilizadas neste trabalho e também uma descrição como é sua utilização, além de descrever os ambientes montados, apresentando cada dispositivo da rede e também detalhando as configurações feitas nos Switches *OpenFlows* e no controlador ONOS .

3.1 Instalação, Interface e Comandos Básicos do ONOS

O ONOS foi o controlador escolhido para ser usado nesse trabalho e já foi apresentado na Seção 2.3. Todas suas documentações, isso inclui instalação, comandos, APIs, etc, podem ser encontrados no site do ONOS¹, porém sua instalação não é algo tão trivial e seguir os tutoriais que há no site do ONOS, não é garantia de êxito na instalação e funcionamento do ONOS, por isso no Apêndice A há um tutorial de instalação que deu certo neste trabalho.

Após a instalação do ONOS, é preciso iniciá-lo, para isso é necessário abrir o terminal do Linux e inserir o seguinte comando²:

```
/opt/onos/bin/onos-service start
```

Se não ocorrer nenhum erro, será mostrado a tela CLI(*Command Line Interface* do ONOS.

O ONOS também oferece uma interface WEB. Para acessar basta abrir o navegador e digitar o endereço **ip_da_máquina:8181/onos/ui/index.html**, irá aparecer a tela de login, em que usuário/senha serão onos/rocks³.

Por meio da interface WEB, é possível enxergar a rede criada, as configurações de cada dispositivo, como também os nós dos *clusters*, caso haja mais de um *cluster*.

Para fins de estudo do ONOS o Mininet⁴ foi usado para emular uma rede e com isso estudar o funcionamento do ONOS.

¹<https://wiki.onosproject.org/display/ONOS/Guides>

²Vale salientar, que a forma de iniciar o ONOS depende de como o mesmo foi instalado, pois há outras formas dele ser instalado. Esse comando funciona com o modo de instalação escolhido para este trabalho.

³É o usuário padrão, para mudar veja a documentação.

⁴Para instalação, acesse o site do Mininet <http://mininet.org/>

A rede que está na Figura 3.1 pode ser montada usando o seguinte comando:

```
sudo mn --controller=remote,ip=ip_da_maquina_onos --topo tree,3
```

Após dar esse comando no terminal do Linux, a rede emulada pelo Mininet irá aparecer⁵ na interface WEB do ONOS, como é mostrado Figura 3.1.

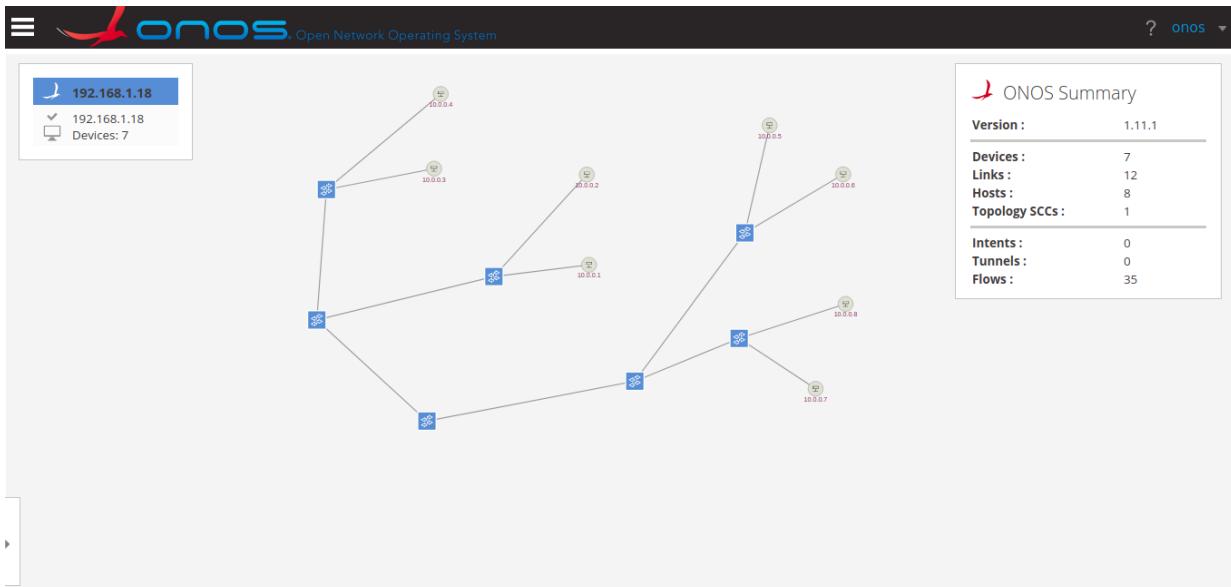


Figura 3.1: Rede Emulada pelo Mininet

Com o entendimento do funcionamento do ONOS, seguiu-se para o próximo passo e na seção 3.2, será mostrado a implantação do Open vSwitch e a criação de um Switch *OpenFlow*.

3.2 Implantação do Switch *OpenFlow* usando o Open vSwitch

Nas Seções 2.5 e 2.6, foram apresentados o *OpenFlow* e Open vSwitch respectivamente. Nesta seção será mostrado a implantação de ambos para a criação de um Switch *OpenFlow*, com o ONOS atuando como controlador.

Neste trabalho, foram usados dois ambientes. Um ambiente foi implementado com uma VM atuando como um Switch *OpenFlow* e em outro ambiente 4 Raspberry Pi atuando como Switch *OpenFlow*.

A instalação do Open vSwitch é simples, basta usar o comando **sudo apt-get install openvswitch-switch**, após a instalação é necessário a criação Switch *OpenFlow*, em ambos ambientes os comandos foram semelhantes, pois os ambientes eram diferentes, logo as configurações deveriam ser diferentes, no Apêndice B tem a sintaxe dos comandos para a criação do Switch *OpenFlow* e a explicação de cada comando. A seguir é apresentado a configuração

⁵Os hosts só aparecem depois de dar o comando *pingall* no CLI do Mininet

na VM criada no primeiro ambiente, e em seguida é mostrado a configuração em um dos Raspberry Pi usados como Siwtch *OpenFlow*:

- VM:

```
ovs-vsctl add-br s1
ifconfig s1 up
ovs-vsctl add-port s1 ens33
ifconfig ens33 0
dhclient s1
ovs-vsctl set-controller s1 tcp:10.11.131.179:6633
```

- Raspberry Pi:

```
ovs-vsctl add-br s1
ifconfig s1 up
ovs-vsctl add-port s1 enxb827eba00f07
ifconfig enxb827eba00f07 0
ifconfig s1 10.1.131.1 netmask 255.255.255.0
ovs-vsctl set-controller s1 tcp:10.1.131.10:6633
```

Apenas com esses comandos é possível criar um Switch *OpenFlow* e permitir que o controlador possa tomar as decisões do plano de controle. Como pode ser observado, e já antes mencionado, houve diferença nas configurações do Switch *OpenFlow* nos diferentes cenários, pois na configuração do switch na VM a interface *Bridge* *s1* recebeu o endereço IP dinamicamente, já no caso do Raspberry, o IP foi setado manualmente, e ambos cenários serão explicados na Subseção 3.3.1 e 3.3.2

Ambas Subseções estão na Seção 3.3, será mostrado o cenário em que foi implantado a rede com um Switch *OpenFlow*, para fins de resultados parciais.

3.3 Rede Montada

Nesta Seção serão descritos ambos cenários montados, tanto o cenário composto por VMs e também o cenário composto pelos Raspberries Pi. Vale salientar que a rede com as VMs foi criado para fins de entender o funcionamento do *Open vSwitch* com o ONOS. Após isso foi implantado o cenário com os Raspberries Pi e mantido o ONOS como controlador.

3.3.1 Rede com VMs

Nesta Seção será descrito o cenário em que foi usado máquinas virtuais em vez do Raspberry Pi, pois em um ambiente virtualizado caso houvesse algum problema, bastaria apenas apagar as máquinas virtuais e fazer outras, caso contrário poderia aplicar o que deu certo em um ambiente real.

Para a criação das máquinas virtuais, foi usado o VMware Workstation 14 Player⁶ e por meio dele foram criados 3 máquinas virtuais, que são:

- onos-tcc-master;

SO: Ubuntu 14.04 LTS Desktop 64 bits

Memória RAM: 2 GB

Processador: AMD A8-5500B APU with Radeon(tm) HD Graphics x 2

HD: 20 GB

IP: 10.1.131.179

ONOS instalado

- open-vswitch-1;

SO: Ubuntu 16.04 Server 32 bits

Memória RAM: 1 GB

Processador: AMD A8-5500B APU with Radeon(tm) HD Graphics x 2

HD: 20 GB

IP: 10.1.131.153

Open vSwitch instalado

- host-1.

SO: Ubuntu 14.04 LTS Desktop 32 bits

Memória RAM: 1 GB

Processador: AMD A8-5500B APU with Radeon(tm) HD Graphics

HD: 20 GB

IP: 10.1.131.169

Apenas o SO instalado

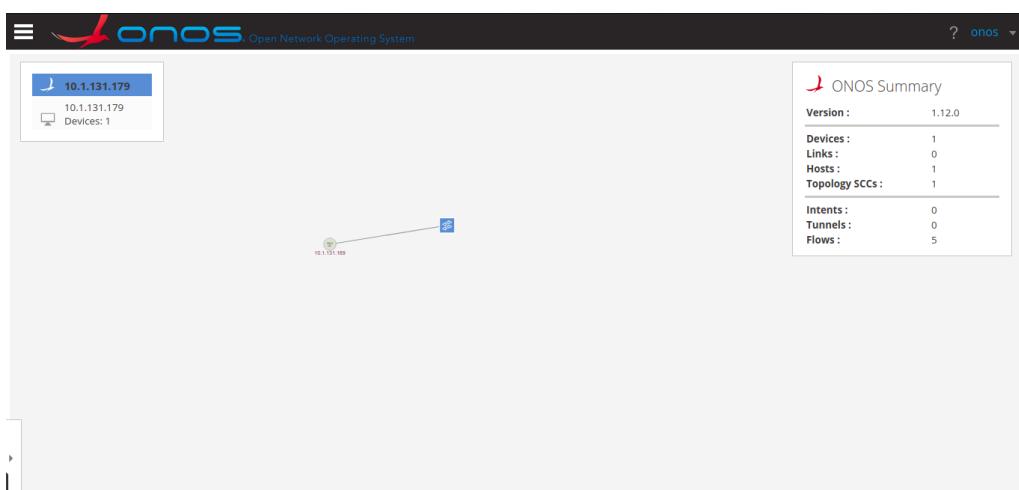


Figura 3.2: Cenário com o Switch OpenFlow criado

⁶Para mais informações <https://pt.wikipedia.org/wiki/VMware>

Todas as máquinas virtuais estavam com a placa de rede virtual configurada em modo *Bridge*, ou seja, as máquinas virtuais receberam IP pela rede física do Laboratório de Redes, que é o local onde foi desenvolvido esse trabalho. Na figura 3.2, é apresentado o ambiente com o Switch *OpenFlow*.

3.3.2 Rede com Raspberry Pi

Nessa Seção será descrito o cenário montado usando 5 Raspberry Pi 3 Model B⁷, o mesmo apresentado na Figura 3.3, sendo 4 deles usados como Switches *OpenFlows* e 1 como Nô IoT.

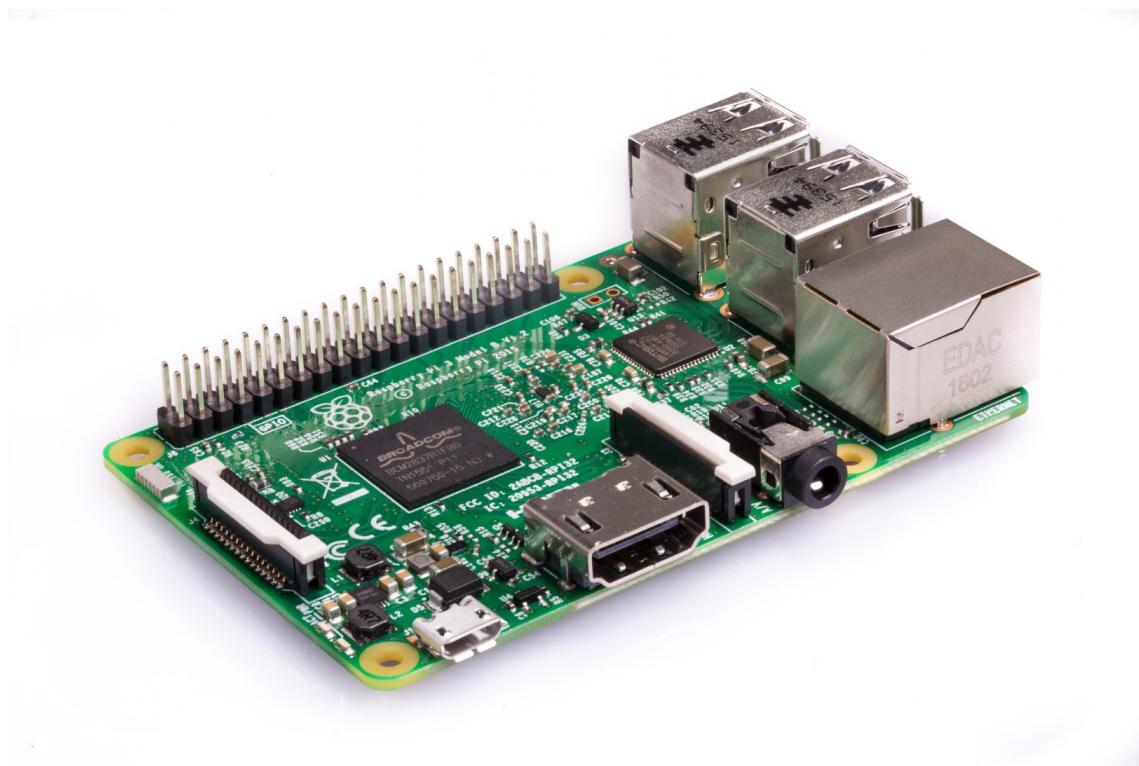


Figura 3.3: *Raspberry Pi 3 Model 3*

Nesse cenário também estão presentes uma VM com o ONOS instalado, outra VM com um servidor DHCP e um Notebook como um *Host*, nesse cenário o servidor DHCP está apenas fornecendo os recursos de IP dinâmico, mas quem controla as configurações de IP do servidor DHCP é o ONOS, ou seja, a VM com o servidor DHCP está apenas com o serviço DHCP ativo, sem configuração alguma e o ONOS é que controla as configurações do DHCP. Para que isso seja possível é necessário ativar a aplicação DHCP do ONOS. Para isso o seguinte comando deve ser dado no CLI do ONOS:

```
app activate org.onosproject.dhcp
```

Após a ativação, é necessário seguir alguns passos. No Apêndice C será detalhado quais passos são esses. Como visto na Seção 3.2 na configuração dos Switches *OpenFlows*, os ender-

⁷Suas especificações podem ser vistas no endereço <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>

reços IPs foram setados manualmente, seguindo a lógica que no último octeto do endereçamento IP fosse referente ao número do switch, e na Tabela 3.1 é apresentado a distribuição IP dos Raspberrys Pi, que estão atuando como Switches *OpenFlows* nesse cenário.

Raspberry	Bridge/Switch	Interface	IP	Máscara
Raspberry 1	s1	enxb827eba00f07	10.1.131.1	255.255.255.0
Raspberry 2	s2	enxb827eb2b751b	10.1.131.2	255.255.255.0
Raspberry 3	s3	enxb827ebb5d6ed	10.1.131.3	255.255.255.0
Raspberry 4	s4	enxb827ebda2307	10.1.131.4	255.255.255.0

Tabela 3.1: Distribuição IP dos Raspberrys Pi

O Raspberry Pi que restou, foi usado como Nô IoT nesse cenário e a sua aplicação IoT era a de um sensor de luminosidade com um motor atuador, que quanto maior a luminosidade maior era o seu ângulo de abertura e quanto menor a luminosidade menor era o seu ângulo de abertura, que variava de 0° à 180°. Níveis de luminosidade foram definidas para alertas, os quais consistiam em acender um led e enviar um arquivo via SSH a um *host* remoto(nesse trabalho era um notebook), indicando o nível de luminosidade. Tais níveis foram definidos da seguinte forma:

- 0% à 25% de luminosidade:
 - Led → Branco;
 - Arquivo → led4.txt;
 - Ângulo → 0° se 0%, caso esteja entre 1% e 25% é 45°.
- 26% à 50% de luminosidade:
 - Led → Vermelho;
 - Arquivo → led3.txt;
 - Ângulo → 90°;
- 51% à 75% de luminosidade:
 - Led → Amarelo;
 - Arquivo → led2.txt;
 - Ângulo → 135°;
- 76% à 100% de luminosidade:
 - Led → Verde;
 - Arquivo → led1.txt;
 - Ângulo → 180°;

No Apêndice D tem o código embarcado da aplicação IoT, a disposição das pinagens no Raspberry e o material usado. Na Figura 3.4 é mostrado como ficou disposto os Raspberries Pi nesse cenário.

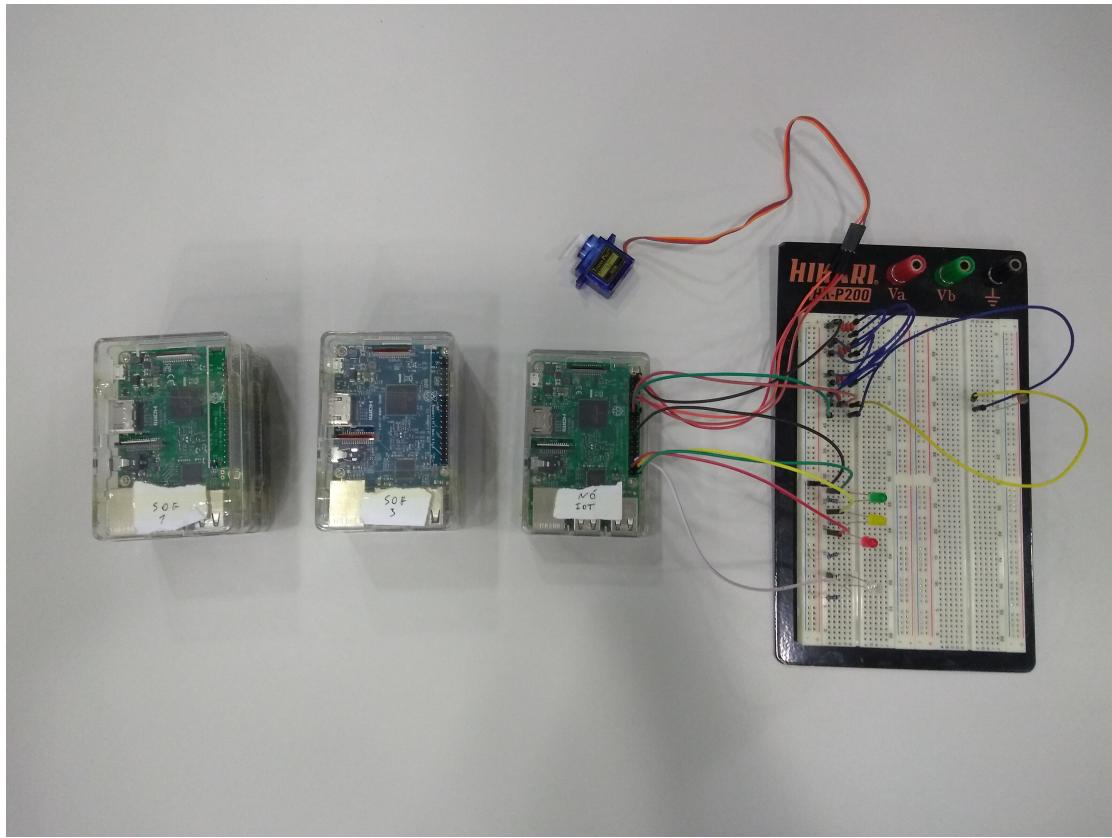


Figura 3.4: Disposição dos Raspberries Pi

Por fim, o ambiente montado com os Raspberries Pi é mostrado na Figura 3.5.

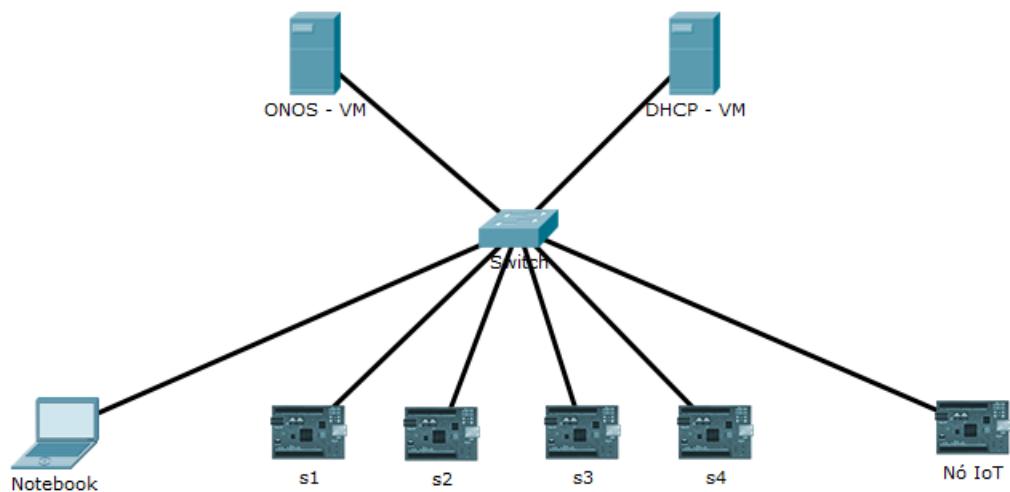


Figura 3.5: Ambiente montado

Como pode ser visto na Figura 3.5, existe um switch. Isso foi necessário, pois os Raspberries Pi que funcionam como Switch *OpenFlow*, tem apenas uma interface de rede cada, sendo

assim não haveria comunicação entre eles e o controlador. Para que houvesse comunicação entre os dispositivos do ambiente montado, foi necessário o uso do switch.

3.4 Mensagens Esperadas

Nessa Seção serão apresentada as mensagens que são esperadas que ocorram, os quais serão apresentadas Seção 4.

A seguir segue a lista das mensagens aguardadas:

- **Three-Way Handshake:** é o processo pelo qual duas máquinas passam para reconhecimento e estabelecimento de conexão entre ambas [David 2011], o processo consiste em 3 mensagens o **SYN**, **SYN-ACK** e **ACK**. O cliente envia um pacote com a flag **SYN** ativa. O servidor responde então com um pacote com as flags **SYN-ACK**. O cliente responde com um pacote **ACK**. Na Figura 3.6 esse processo é apresentado.

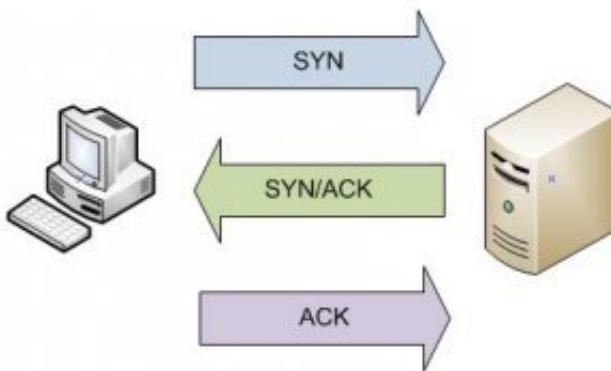


Figura 3.6: Three-Way Handshake

- **OFPT_HELLO:** É usado por qualquer controlador ou switch durante a configuração da conexão, em que é feita a negociação de versão. No momento do estabelecimento da conexão, cada lado deve enviar imediatamente uma mensagem Hello com o campo de versão configurado para a versão mais alta suportada pelo remetente. Se a negociação falhar, uma mensagem de erro é enviada com o tipo *HelloFailed* [Flowgrammable 2012]. Na Figura 3.7 esse processo pode ser observado.
- **OFPT_FEATURES_REQUEST** e **OFPT_FEATURES_REPLY**: Quando um canal de transporte (TCP, SCTP, TLS) é estabelecido, a primeira atividade é a determinação do recurso. O controlador enviará um *FeatureReq*. O pedido de recurso consiste apenas em uma mensagem de cabeçalho *OpenFlow*, com o valor *FeatureReq*. O switch responderá a essa solicitação com suas capacidades na mensagem *FeatureReply*. Esta sequência básica não muda em versões *OpenFlow*. Na Figura 3.8
- **OFPT_MULTIPART_REQUEST**(*MultipartReq*): É usado para solicitar informações sobre fluxos individuais. Na versão 1.3 essa mensagem recebeu o nome de

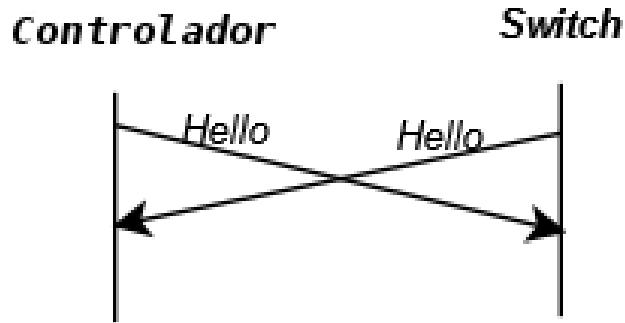


Figura 3.7: Mensagens `OFPT_HELLO`. Adaptado de [Flowgrammable 2012]

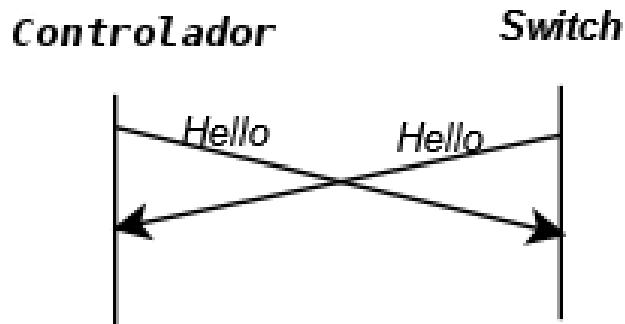


Figura 3.8: Mensagens `OFPT_FEATURES_REQUEST` e `REPLY`. Adaptado de [Flowgrammable 2012]

MultipartReq, anteriormente chamado de **StatsReq**. *Port Type* foi renomeado para ser **PortStats**. Além disso, novos campos foram adicionados. Na Figura 3.9 é apresentado o processo de troca de mensagem e na Figura 3.10 é mostrado o pacote.

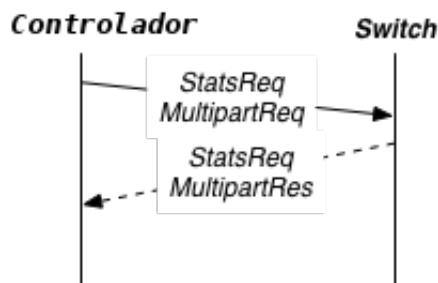


Figura 3.9: Mensagem `OFPT_MULTIPART_REQUEST`. Adaptado de [Flowgrammable 2012]

- **OFPT_MULTIPART_REPLY**(*MultipartRes*: *MultipartRes* é a resposta ao *MultipartReq*.

Na Figura 3.11 é apresentado o processo de resposta ao *MultipartReq* e na Figura 3.12 é mostrado o pacote.

- **Mensagens DHCP**: Como mostrado na Figura 3.13, quando um dispositivo configurado com DHCP se conecta à rede, o cliente transmite uma mensagem de descoberta DHCP (**DHCPODISCOVER**) para identificar qualquer servidor DHCP na rede. Um servidor DHCP responde com uma mensagem de pacote DHCP (**DHCPOFFER**),

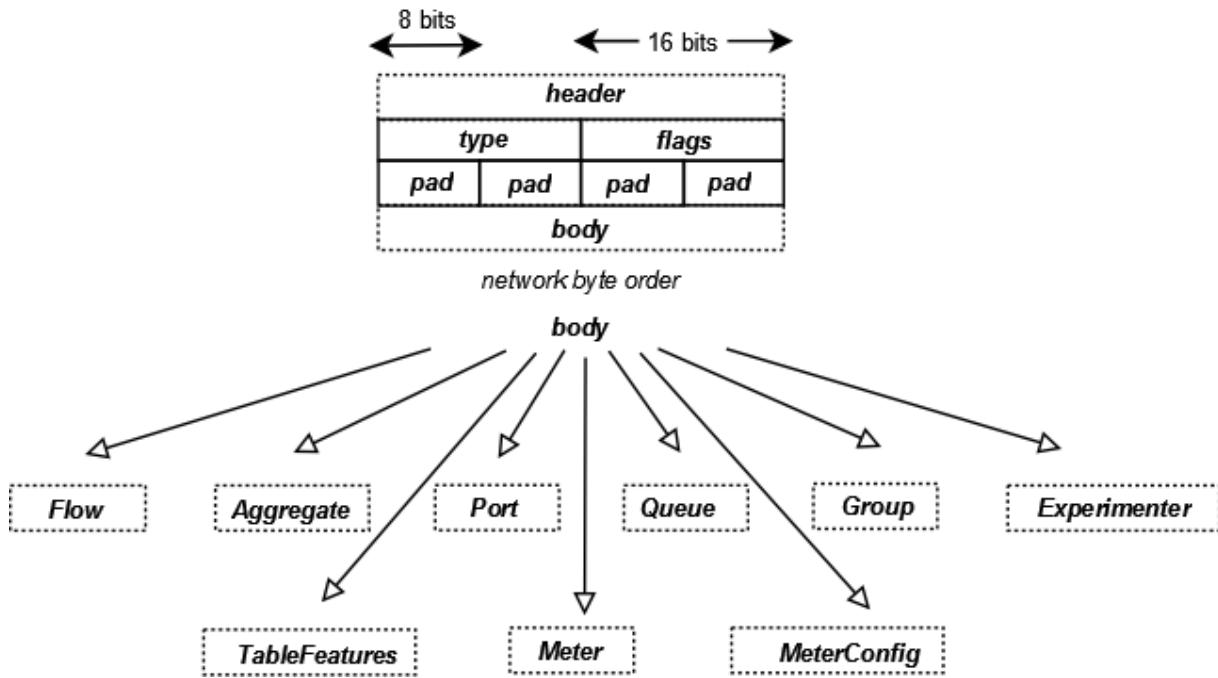


Figura 3.10: Pacote `OFPT_MULTIPART_REQUEST`. Fonte [Flowgrammable 2012]

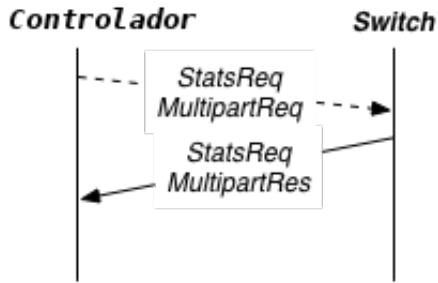


Figura 3.11: Mensagem `OFPT_MULTIPART_REPLY`. Adaptado de [Flowgrammable 2012]

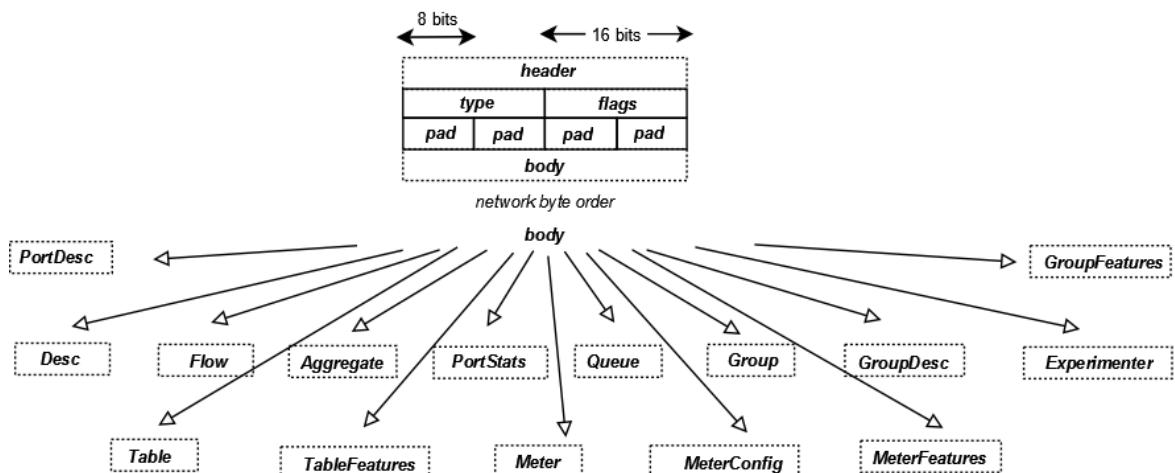


Figura 3.12: Pacote `OFPT_MULTIPART_REPLY`. Fonte [Flowgrammable 2012]

que oferece um *leasing* ao cliente. A mensagem de oferta contém o endereço IP e a máscara de sub-rede a serem atribuídos, além do endereço do servidor DNS e o endereço do *gateway* padrão, após isso o cliente transmitir uma mensagem de solicitação

de DHCP (**DHCPREQUEST**) e a oferta que o cliente está aceitando. Um cliente pode decidir solicitar um endereço que já havia sido alocado pelo servidor, se o endereço IP solicitado pelo cliente, ainda seja válido, o servidor retornará uma mensagem de confirmação DHCP (**DHCPACK**) que confirma para o cliente que o processo foi finalizado.



Figura 3.13: Processo DHCP. Retirado de [CCNA 2013]

Na Seção 4, serão apresentados e explicados os resultados obtidos.

Capítulo 4

Resultados

Para obtenção dos resultados finais, foi usado em ambos cenários citados nas Seções 3.3.1 e 3.3.2 o *Wireshark*, que é um programa que analisa o tráfego de rede, e o organiza por protocolos [Brito 2014]. Porém como o foco do trabalho é no uso dos Raspberrys, do cenário virtual, só será apresentado a captura dos pacotes feita durante as primeiras mensagens trocadas entre o Switch *OpenFlow* e o controlador, para isso o *Wireshark* foi ativado antes do Switch *OpenFlow* ser criado na máquina virtual, para que os primeiros pacotes pudessem ser capturados pelo *Wireshark*. Na Figura 4.1 é apresentado a captura obtida com a filtragem de pacotes TCP na porta 6633 que é a porta padrão *OpenFlow*, e das flags SYN, SYN-ACK e ACK, tal filtragem pode ser feita com a seguinte expressão¹:

```
tcp.port == 6633 && tcp.flags.ack || tcp.flags.syn
```

Como pode ser observado na Figura 4.1, os primeiros pacotes são do *Three-Way Handshake* do protocolo TCP, e como esperado, após isso vem a mensagem OFPT_HELLO. Após esse reconhecimento entre Switch e controlador, esperasse que o controlador envie a mensagem OFPT_FEATURES_REQUEST ao Switch. Como é visto na Figura 4.2 isso ocorre, pois agora o controlador precisa saber quais recursos o Switch *OpenFlow* tem disponível, para isso o switch enviará a mensagem OFPT_FEATURES_REPLY².

Na resposta, o switch *OpenFlow* enviará suas características, como por exemplo o *n_buffers*, que é a capacidade de quantos pacotes podem compor a fila. Na Figura 4.2 é mostrado a mensagem OFPT_FEATURES_REPLY do Switch *OpenFlow*. Observa-se três informações importantes nessa resposta, o *datapath_id*, *n_tables* e *capabilities*.

O *datapath_id* possui o identificador do Switch da rede que é único, pois é baseado no endereço MAC do switch. O campo *n_tables* indica que o switch pode ter até 254 tabelas, e por fim é apresentado o *capabilities*, que está indicando todas as capacidades que o Switch tem, e isso é indicado onde tiver o *True*, se for *False* significa que tal função está não está disponível. Na Tabela 4.1 contém a descrição de cada *capabilities*.

¹Todas as capturas estarão disponíveis no seguinte repositório Git <https://github.com/joabsilva/TCC>

²Todos os detalhes de cada mensagem *OpenFlow*, pode ser encontrado nesse PDF <https://goo.gl/55NuA1>

```

▶ Frame 32792: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▶ Ethernet II, Src: VMware_07:79:56 (00:0c:29:07:79:56), Dst: VMware_ac:97:cb (00:0c:29:ac:97:cb)
▶ Internet Protocol Version 4, Src: 10.1.131.153, Dst: 10.1.131.179
▼ Transmission Control Protocol, Src Port: 41372, Dst Port: 6633, Seq: 0, Len: 0
  Source Port: 41372
  Destination Port: 6633
  [Stream index: 7]
  [TCP Segment Len: 0]
  Sequence number: 0 (relative sequence number)
  Acknowledgment number: 0
  Header Length: 40 bytes
  ▶ Flags: 0x002 (SYN)
  Window size value: 29200
  [Calculated window size: 29200]
  Checksum: 0x3316 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  ▶ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
  ▶ [SEQ/ACK analysis]

▶ Frame 32794: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▶ Ethernet II, Src: VMware_ac:97:cb (00:0c:29:ac:97:cb), Dst: VMware_07:79:56 (00:0c:29:07:79:56)
▶ Internet Protocol Version 4, Src: 10.1.131.179, Dst: 10.1.131.153
▼ Transmission Control Protocol, Src Port: 6633, Dst Port: 41372, Seq: 0, Ack: 1, Len: 0
  Source Port: 6633
  Destination Port: 41372
  [Stream index: 7]
  [TCP Segment Len: 0]
  Sequence number: 0 (relative sequence number)
  Acknowledgment number: 1 (relative ack number)
  Header Length: 40 bytes
  ▶ Flags: 0x012 (SYN, ACK)
  Window size value: 28960
  [Calculated window size: 28960]
  Checksum: 0xecc0 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  ▶ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
  ▶ [SEQ/ACK analysis]

▶ Frame 32794: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▶ Ethernet II, Src: VMware_ac:97:cb (00:0c:29:ac:97:cb), Dst: VMware_07:79:56 (00:0c:29:07:79:56)
▶ Internet Protocol Version 4, Src: 10.1.131.179, Dst: 10.1.131.153
▼ Transmission Control Protocol, Src Port: 6633, Dst Port: 41372, Seq: 0, Ack: 1, Len: 0
  Source Port: 6633
  Destination Port: 41372
  [Stream index: 7]
  [TCP Segment Len: 0]
  Sequence number: 0 (relative sequence number)
  Acknowledgment number: 1 (relative ack number)
  Header Length: 40 bytes
  ▶ Flags: 0x012 (SYN, ACK)
  Window size value: 28960
  [Calculated window size: 28960]
  Checksum: 0xecc0 [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
  ▶ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
  ▶ [SEQ/ACK analysis]

▶ Frame 32799: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▶ Ethernet II, Src: VMware_07:79:56 (00:0c:29:07:79:56), Dst: VMware_ac:97:cb (00:0c:29:ac:97:cb)
▶ Internet Protocol Version 4, Src: 10.1.131.153, Dst: 10.1.131.179
▶ Transmission Control Protocol, Src Port: 41372, Dst Port: 6633, Seq: 1, Ack: 1, Len: 8
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_HELLO (0)
  Length: 8
  Transaction ID: 13

```

Figura 4.1: Captura do Handshake TCP e a primeira mensagem OpenFlow

capabilities	Descrição
OFPC_FLOW_STATS	Estatísticas do fluxo
OFPC_TABLE_STATS	Estatísticas da tabela
OFPC_PORT_STATS	Estatísticas da porta
OFPC_GROUP_STATS	Estatísticas de grupo
OFPC_IP_REASM	Pode montar fragmentos de IP
OFPC_QUEUE_STATS	Estatísticas da fila
OFPC_PORT_BLOCKED	Switch irá bloquear as portas em loop

Tabela 4.1: Descrição das capacidades do Switch OpenFlow 1.3

```

▶ Frame 32945: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
▶ Ethernet II, Src: Vmware_ac:97:cb (00:0c:29:ac:97:cb), Dst: Vmware_07:79:56 (00:0c:29:07:79:56)
▶ Internet Protocol Version 4, Src: 10.1.131.179, Dst: 10.1.131.153
▶ Transmission Control Protocol, Src Port: 6633, Dst Port: 41374, Seq: 17, Ack: 9, Len: 8
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_FEATURES_REQUEST (5)
  Length: 8
  Transaction ID: 4294967294

▶ Frame 32949: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
▶ Ethernet II, Src: Vmware_07:79:56 (00:0c:29:07:79:56), Dst: Vmware_ac:97:cb (00:0c:29:ac:97:cb)
▶ Internet Protocol Version 4, Src: 10.1.131.153, Dst: 10.1.131.179
▶ Transmission Control Protocol, Src Port: 41374, Dst Port: 6633, Seq: 9, Ack: 25, Len: 32
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_FEATURES_REPLY (6)
  Length: 32
  Transaction ID: 4294967294
  datapath_id: 0x0000000c29077956
  n_buffers: 256
  n_tables: 254
  auxiliary_id: 0
  Pad: 0
  ▼ capabilities: 0x00000004f
    .... .... .... .... .... .1 = OFPC_FLOW_STATS: True
    .... .... .... .... ..1. = OFPC_TABLE_STATS: True
    .... .... .... .... .1.. = OFPC_PORT_STATS: True
    .... .... .... .... 1... = OFPC_GROUP_STATS: True
    .... .... .... .... .0. .... = OFPC_IP_REASM: False
    .... .... .... .... 1.. .... = OFPC_QUEUE_STATS: True
    .... .... .... .... .0 .... .... = OFPC_PORT_BLOCKED: False
  Reserved: 0x000000000

```

Figura 4.2: Requisição dos recursos do Switch OpenFlow e de resposta do Switch

Para fins de comparação com os pacotes capturados, a Figura 4.3 apresenta as configurações do Switch OpenFlow.

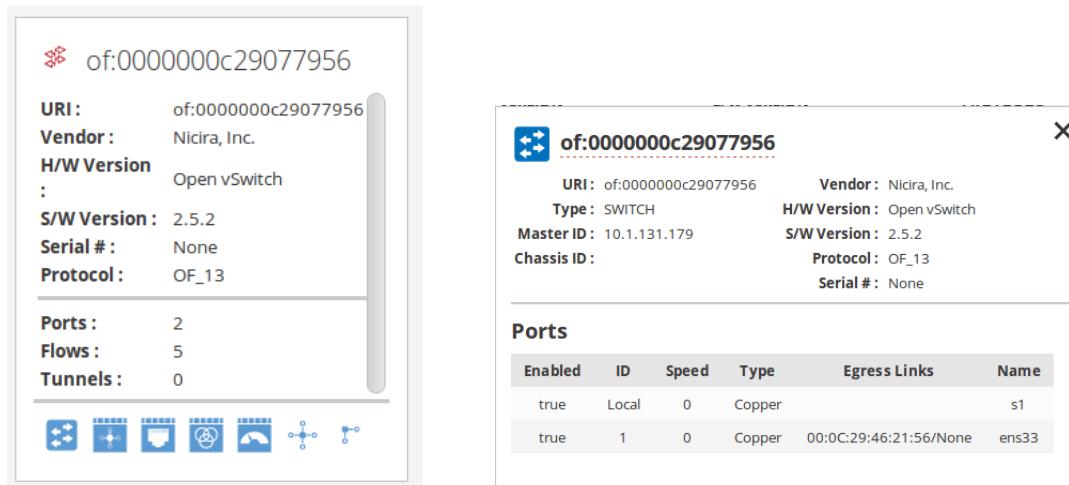


Figura 4.3: Capacidades do Switch OpenFlow

Visto que o cenário com VMs implementa SDN, pôde-se trabalhar com a rede com os Raspberries Pi, vale salientar que como na rede virtual, as primeiras mensagens *OpenFlows* foram as mesmas. A partir daqui será apresentado os resultados obtidos na rede com os Raspberries após as mensagens anteriormente citadas, as capturas que serão mostrada, foram feitas no servidor em que estava instalado o ONOS e analisando os pacotes vindo do Switch OpenFlow s1(ver Tabela 3.1).

Após as primeiras mensagens trocadas entre o controlador e o Switch, como esperado a próxima mensagem foi a OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC e em seguida OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC e como pode ser visto na

Figura 4.4, está de acordo com o que foi explicado na Seção 3.4.

```

▶ Frame 187: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
▶ Ethernet II, Src: PcsCompu_40:27:ac (08:00:27:40:27:ac), Dst: Raspberr_a0:0f:07 (b8:27:eb:a0:0f:07)
▶ Internet Protocol Version 4, Src: 10.1.131.10, Dst: 10.1.131.1
▶ Transmission Control Protocol, Src Port: 6633, Dst Port: 48180, Seq: 25, Ack: 41, Len: 16
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REQUEST (18)
  Length: 16
  Transaction ID: 4294967293
  Type: OFPMP_PORT_DESC (13)
  Flags: 0x0000
  Pad: 00000000

▶ Frame 188: 210 bytes on wire (1680 bits), 210 bytes captured (1680 bits)
▶ Ethernet II, Src: Raspberr_a0:0f:07 (b8:27:eb:a0:0f:07), Dst: PcsCompu_40:27:ac (08:00:27:40:27:ac)
▶ Internet Protocol Version 4, Src: 10.1.131.1, Dst: 10.1.131.10
▶ Transmission Control Protocol, Src Port: 48180, Dst Port: 6633, Seq: 41, Ack: 41, Len: 144
▼ OpenFlow 1.3
  Version: 1.3 (0x04)
  Type: OFPT_MULTIPART_REPLY (19)
  Length: 144
  Transaction ID: 4294967293
  Type: OFPMP_PORT_DESC (13)
  Flags: 0x0000
  Pad: 00000000
  ▼ Port
    Port no: OFPP_LOCAL (0xffffffff)
    Pad: 00000000
    Hw addr: Raspberr_a0:0f:07 (b8:27:eb:a0:0f:07)
    Pad: 0000
    Name: s1
    Config: 0x00000000
    State: 0x00000000
    Current: 0x00000000
    Advertised: 0x00000000
    Supported: 0x00000000
    Peer: 0x00000000
    Curr speed: 0
    Max speed: 0
  ▼ Port
    Port no: 1
    Pad: 00000000
    Hw addr: Raspberr_a0:0f:07 (b8:27:eb:a0:0f:07)
    Pad: 0000
    Name: enxb827eba00f07
    Config: 0x00000000
    State: 0x00000000
    Current: 0x00002008
    Advertised: 0x0000e80f
    Supported: 0x0000280f
    Peer: 0x00000000
    Curr speed: 100000
    Max speed: 100000

```

Figura 4.4: Mensagens *OFPT_MULTIPART_REQUEST* e *REPLY*

Como pode ser observado, as informações das portas disponíveis estão presente, tanto a Bridge **s1** quanto a interface física **enxb827eba00f07**, além de informações como MAC e número da porta.

Após analisar as mensagens iniciais entre controlador e Switch *OpenFlow*, será apresentado as capturas feitas durante as requisições de endereço IP no nó IoT. Para isso a seguinte filtragem foi usada:

```
udp.port == 68 || udp.port == 67
```

Como falado na Seção 3.4, as mensagens esperadas durante a requisição DHCP são **Discovery**, **Request**, **Offer** e **ACK**, e como pode ser observado na Figura 4.5 temos a sequência de mensagens e em destaque a mensagem **ACK** com o IP fornecido pelo servidor DHCP, dentro dos parâmetros de rede determinado pelo controlador ONOS.

Essas mesmas mensagens podem também ser observadas nos Switches *OpenFlow*, usando novamente o Switch **s1** como exemplo, pode-se observar a mesma mensagem DHCP **ACK**

1035	304.8128977...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover	- Transaction ID 0xfde9a90d
1043	306.5546331...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover	- Transaction ID 0xc9e784f
1065	309.4968568...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover	- Transaction ID 0xc9e784f
1087	315.9136981...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover	- Transaction ID 0xc9e784f
1088	315.9766668...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request	- Transaction ID 0xc9e784f
1089	316.0094351...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover	- Transaction ID 0x870fb4d
1090	316.0166588...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request	- Transaction ID 0x870fb4d
1112	319.2930873...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover	- Transaction ID 0xfde9a90d
1115	319.3135077...	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer	- Transaction ID 0xfde9a90d
1116	319.3138142...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request	- Transaction ID 0xfde9a90d
1117	319.3207222...	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer	- Transaction ID 0xfde9a90d
1118	319.3235188...	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer	- Transaction ID 0xfde9a90d
1119	319.3239343...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1120	319.3248211...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1121	319.3257572...	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer	- Transaction ID 0xfde9a90d
1122	319.3265893...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1123	319.3284250...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1135	322.8997454...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request	- Transaction ID 0xfde9a90d
1136	322.9061249...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1137	322.9063343...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1138	322.9081097...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1139	322.9090268...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1153	326.9673065...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request	- Transaction ID 0xfde9a90d
1154	326.9749714...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1155	326.9763954...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1156	326.9766409...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1157	326.9766833...	10.1.11.50	10.1.11.56	DHCP	342	DHCP NAK	- Transaction ID 0xfde9a90d
1195	337.6370051...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover	- Transaction ID 0xebf44f08
1196	337.6446858...	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer	- Transaction ID 0xebf44f08
1197	337.6450468...	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request	- Transaction ID 0xebf44f08
1198	337.6457256...	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer	- Transaction ID 0xebf44f08
1199	337.6463024...	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer	- Transaction ID 0xebf44f08
1200	337.6478886...	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer	- Transaction ID 0xebf44f08
1201	337.6528657...	10.1.11.50	255.255.255.255	DHCP	342	DHCP ACK	- Transaction ID 0xebf44f08
1202	337.6529479...	10.1.11.50	10.1.11.55	DHCP	342	DHCP NAK	- Transaction ID 0xebf44f08
1203	337.6620020...	10.1.11.50	10.1.11.55	DHCP	342	DHCP NAK	- Transaction ID 0xebf44f08

Frame 1201: 342 bytes on wire (2736 bits), 342 bytes captured (2736 bits) on interface 0
 ▶ Ethernet II, Src: ca:fe:ca:fe:ca:fe (ca:fe:ca:fe:ca:fe), Dst: Raspberry_2b:e0:c8 (b8:27:eb:2b:e0:c8)
 ▶ Internet Protocol Version 4, Src: 10.1.11.50, Dst: 255.255.255.255
 ▶ User Datagram Protocol, Src Port: 67, Dst Port: 68
 ▶ Bootstrap Protocol (ACK)
 Message type: Boot Reply (2)
 Hardware type: Ethernet (0x01)
 Hardware address length: 6
 Hops: 0
 Transaction ID: 0xebf44f08
 Seconds elapsed: 0
 Bootp flags: 0x0000 (Unicast)
 Client IP address: 0.0.0.0
 Your (client) IP address: 10.1.11.55
 Next server IP address: 10.1.11.50
 Relay agent IP address: 0.0.0.0
 Client MAC address: Raspberry_2b:e0:c8 (b8:27:eb:2b:e0:c8)
 Client hardware address padding: 00000000000000000000
 Server host name not given
 Boot file name not given
 Magic cookie: DHCP
 ▶ Option: (53) DHCP Message Type (ACK)
 ▶ Option: (54) DHCP Server Identifier
 ▶ Option: (51) IP Address Lease Time
 ▶ Option: (58) Renewal Time Value
 ▶ Option: (59) Rebinding Time Value
 ▶ Option: (1) Subnet Mask
 ▶ Option: (28) Broadcast Address
 ▶ Option: (3) Router
 ▶ Option: (6) Domain Name Server
 ▶ Option: (255) End
 Padding: 0000000000000000

Figura 4.5: Captura das mensagens DHCP

da Figura 4.5, mas com um acréscimo, o protocolo *OpenFlow* adiciona em seu cabeçalho a mensagem DHCP, como pode ser visto na Figura 4.6

No Capítulo 5, os resultados obtidos serão discutidos.

No.	Time	Source	Destination	Protocol	Length	Info
4108	334.711980	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer - Transaction ID 0xb06dce44
4109	334.713199	10.1.131.1	10.1.131.10	OpenFlow	450	Type: OFPT_PACKET_IN
4110	334.724985	10.1.131.10	10.1.131.1	OpenFlow	448	Type: OFPT_PACKET_OUT
4111	334.725465	10.1.11.50	10.1.11.59	DHCP	342	DHCP NAK - Transaction ID 0xb06dce44
4163	335.945372	10.1.131.1	10.1.131.10	OpenFlow	450	Type: OFPT_PACKET_IN
4164	335.950263	10.1.131.10	10.1.131.1	OpenFlow	448	Type: OFPT_PACKET_OUT
4166	335.950732	10.1.11.50	255.255.255.255	DHCP	342	DHCP ACK - Transaction ID 0xebf44f08
6274	483.329148	10.1.131.1	10.1.131.10	OpenFlow	450	Type: OFPT_PACKET_IN
6275	483.332615	10.1.131.10	10.1.131.1	OpenFlow	448	Type: OFPT_PACKET_OUT
6277	483.333114	10.1.11.50	255.255.255.255	DHCP	342	DHCP ACK - Transaction ID 0xebf44f08
6337	486.735897	10.1.131.1	10.1.131.10	OpenFlow	450	Type: OFPT_PACKET_IN
6338	486.744126	10.1.131.10	10.1.131.1	OpenFlow	448	Type: OFPT_PACKET_OUT
6340	486.744411	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer - Transaction ID 0x2bd80278
6341	486.747294	10.1.131.1	10.1.131.10	OpenFlow	450	Type: OFPT_PACKET_IN
6342	486.752761	10.1.131.10	10.1.131.1	OpenFlow	448	Type: OFPT_PACKET_OUT
6343	486.752993	10.1.11.50	10.1.11.59	DHCP	342	DHCP NAK - Transaction ID 0x2bd80278
6345	486.786606	10.1.131.1	10.1.131.10	OpenFlow	450	Type: OFPT_PACKET_IN
6346	486.792757	10.1.131.10	10.1.131.1	OpenFlow	448	Type: OFPT_PACKET_OUT
6348	486.793036	10.1.11.50	255.255.255.255	DHCP	342	DHCP Offer - Transaction ID 0x8a8dc677
6349	486.797201	10.1.131.1	10.1.131.10	OpenFlow	450	Type: OFPT_PACKET_IN
6350	486.802098	10.1.131.10	10.1.131.1	OpenFlow	448	Type: OFPT_PACKET_OUT
6351	486.802319	10.1.11.50	10.1.11.59	DHCP	342	DHCP NAK - Transaction ID 0x8a8dc677
8267	617.266362	10.1.131.1	10.1.131.10	OpenFlow	450	Type: OFPT_PACKET_IN
8268	617.282312	10.1.131.1	10.1.131.10	OpenFlow	450	Type: OFPT_PACKET_IN

► Frame 4164: 448 bytes on wire (3584 bits), 448 bytes captured (3584 bits)
 ► Ethernet II, Src: PcsCompu_40:27:ac (08:00:27:40:27:ac), Dst: Raspberry_a0:0f:07 (b8:27:eb:a0:0f:07)
 ► Internet Protocol Version 4, Src: 10.1.131.10, Dst: 10.1.131.1
 ► Transmission Control Protocol, Src Port: 6633, Dst Port: 37768, Seq: 94372, Ack: 612952, Len: 382
 ► OpenFlow 1.3
 ► OpenFlow 1.3
 Version: 1.3 (0x04)
 Type: OFPT_PACKET_OUT (13)
 Length: 382
 Transaction ID: 14537
 Buffer ID: OFPP_NO_BUFFER (0xffffffff)
 In port: OFPP_CONTROLLER (0xfffffffffd)
 Actions length: 16
 Pad: 000000000000
 ► Action
 ► Data
 ► Ethernet II, Src: ca:fe:ca:fe:ca:fe (ca:fe:ca:fe:ca:fe), Dst: Raspberry_2b:e0:c8 (b8:27:eb:2b:e0:c8)
 ► Internet Protocol Version 4, Src: 10.1.11.50, Dst: 255.255.255.255
 ► User Datagram Protocol, Src Port: 67, Dst Port: 68
 ► Bootstrap Protocol (ACK)
 Message type: Boot Reply (2)
 Hardware type: Ethernet (0x01)
 Hardware address length: 6
 Hops: 0
 Transaction ID: 0xebf44f08
 Seconds elapsed: 0
 ► Boot flags: 0x0000 (Unicast)
 Client IP address: 0.0.0.0
 Your (client) IP address: 10.1.11.55
 Next server IP address: 10.1.11.50
 Relay agent IP address: 0.0.0.0
 Client MAC address: Raspberry_2b:e0:c8 (b8:27:eb:2b:e0:c8)
 Client hardware address padding: 00000000000000000000
 Server host name not given
 Boot file name not given
 Magic cookie: DHCP
 Option: (53) DHCP Message Type (ACK)
 Option: (54) DHCP Server Identifier
 Option: (51) IP Address Lease Time
 Option: (58) Renewal Time Value
 Option: (59) Rebinding Time Value
 Option: (1) Subnet Mask
 Option: (28) Broadcast Address
 Option: (3) Router
 Option: (6) Domain Name Server
 Option: (255) End
 Padding: 0000000000000000

Figura 4.6: Captura da mensagem DHCP ACK no switch 1(s1)

Capítulo 5

Discussão dos Resultados

Como pôde ser visto, é possível usar o Raspberry Pi como Switch *OpenFlow*, visto que as mensagens *OpenFlow* esperadas foram capturadas durante o processo de inicialização da comunicação entre controlador e Switch *OpenFlow*. Há algumas ressalvas que devem ser mencionadas: há a questão da limitação de interfaces de rede, pois como o Raspberry tem apenas uma interface de rede, então foi necessário a utilização de um Switch para a conexão entre os Raspberries Pi(switches *OpenFlows*) e controlador, e também conectar o Nô IoT à rede, porém o ideal seria que o Nô IoT deveria estar conectado diretamente ao Raspberry, para que todo tráfego gerado por ele, passasse pelo o Raspberry o qual tivesse conectado.

Mesmo com essa limitação, no geral, houve êxito no objetivo desse trabalho, pois mesmo que todo o tráfego gerado pelo Nô IoT não passasse pelos Raspberries Pi, tanto a aplicação IoT explicado na Seção 3.3, quanto a comunicação *OpenFlow* e o serviço DHCP funcionaram como esperado.

Ao usar a aplicação DHCP no ONOS, o SDN pôde controlar os recursos que seriam disponibilizados ao Nô IoT, com isso pôde-se fazer as capturas para análise de como o controlador irá atuar sobre o Nô IoT. As mensagens DHCP ocorreram como era previsto, porém é bom ressaltar duas coisas observadas durante as capturas das mensagens DHCPs. Uma é a mensagem **NAK** aparecendo repetidas vezes como pôde ser vista na Figura 4.5, e esse comportamento também foi observado no *Host*. A mensagem **NAK** ocorre quando a oferta daquele IP não é mais válida, talvez devido a um tempo limite ou a outro cliente que tenha o mesmo endereço [CCNA 2013], porém é de se estranhar que uma rede com apenas dois Nós que requisitam o endereço IP via DHCP, ocorra esse tipo de comportamento. A outra coisa que vale ressaltar, é o número excessivo de mensagens **Discovery**, o que em uma rede comum não é comum ver tantas mensagens desse tipo partindo de um dispositivo. Até o presente momento desse trabalho não foi encontrado os motivos de tais comportamentos. Apesar desse comportamento durante a requisição DHCP, o nô IoT e o *host* receberam IP dinamicamente.

Capítulo 6

Considerações Finais

Após tudo o que foi apresentado neste trabalho, pode-se dizer que apesar das questões ditas sobre o DHCP, no geral é totalmente possível usar Raspberries Pi como Switch *OpenFlow* em uma rede SDN, e também foi observado que mesmo no cenário montado para esse trabalho, o ONOS proveu recursos ao Nô IoT, disponibilizando endereçamento IP ao mesmo por meio da aplicação DHCP do ONOS.

Os Raspberries também são soluções econômicas, visto que os valores de Switches com suporte *OpenFlow* são altos¹, podendo girar em torno de R\$ 4.000,00, e apesar da limitação de portas *Ethernet* isso pode ser resolvido com adaptadores USB-*Ethernet*² ou usar uma placa Zodiac FX³, que são soluções com preços bons.

Além disso o Raspberry dependendo da necessidade do ambiente que se queira implantá-lo, o Raspberry pode ter função de Nô IoT ao mesmo tempo que serve como Switch *OpenFlow*, assim permitindo ter um dispositivo que pode enviar mensagens ou alertas de temperatura ou umidade para um servidor remoto, por exemplo.

Somando a tudo isso, a nível de controlador, há projeto com o ONOS em ambientes empresariais, em que fornece recursos como reserva de pool de recursos de rede sob demanda, implantação automática(*Automatic Deployment*), otimização inteligente e ajuste de largura de banda sob demanda para clientes corporativos em campi, operadoras e redes de data center [ONOS 2016], permitindo assim uma rede flexível para testes os quais podem ser incluídos aplicações IoT.

¹Segue uma pesquisa feita de valores <https://goo.gl>AnKroP>

²<https://goo.gl/cqwzHB>

³<https://northboundnetworks.com/products/zodiac-fx>

Capítulo 7

Sugestões para trabalhos futuros

Para trabalhos futuros pretende-se aplicar o que não foi aplicado nesse trabalho, como o encaminhamento do tráfego gerado pelo Nô IoT, além disso acrescentar mais Nôs IoT na rede.

Também tentar obter adaptadores USB-*Ethernet*, para não precisar mais usar um switch.

Buscar também entender o comportamento das mensagens DHCP desse trabalho.

Com esses objetivos pretende-se produzir artigos para eventos e periódicos.

Apêndice A

Instalação ONOS

Essa instalação pode ser encontrada no seguinte Link <https://goo.gl/apLRQf>

A.1 Pré-requisitos

Pré-requisitos de máquina

- 2 Núcleos de CPU
- 2 GB RAM
- 10 GB hdd
- 1 interface de rede de qualquer velocidade

Pré-requisitos de porta. Essas portas já são abertas automaticamente.

- 8181 para REST API e GUI
- 8101 para acesso ao ONOS CLI
- 9876 para comunicação entre clusters
- 6633 para OpenFlow (No site do ONOS é dito 6653, mas aqui a portão padrão *OpenFlow* funcionou)
- 6640 para o OVSDB

O ONOS é uma plataforma baseada em Java, então deve-se instalar o Java 8

```
sudo apt-get install software-properties-common -y && \
sudo add-apt-repository ppa:webupd8team/java -y && \
sudo apt-get update && \
echo "oracle-java8-installer shared/accepted-oracle-license-v1-1
select true" | sudo debconf-set-selections && \
sudo apt-get install oracle-java8-installer oracle-java8-set-default
-y
```

Instalar o Curl

```
sudo apt-get install curl
```

Caso não haja essa pasta em seu PC, crie ele.

```
sudo mkdir /opt
cd /opt
```

A.2 Instalação

Verifique a versão do ONOS que você deseja e no lugar de \$ONOS_VERSION coloque a versão desejada, recomendo usar a versão 1.12.1, pois é a usada nesse trabalho, caso queira outra versão segue o link <https://goo.gl/ekLzpd> e faça o download

```
sudo wget -c http://downloads.onosproject.org/release/onos-
$ONOS_VERSION.tar.gz
```

Extraia o arquivo para a pasta /opt

```
sudo tar xzf onos-$ONOS_VERSION.tar.gz
```

Altere o nome da pasta

```
sudo mv onos-$ONOS_VERSION onos
```

Após isso é só rodar o script de inicialização do ONOS

```
sudo /opt/onos/bin/onos-service start
```

Após iniciar, no terminal deve aparecer a tela que aparece na Figura A.1.

```
onos@onos-tcc-master:~$ /opt/onos/bin/onos-service start
Welcome to Open Network Operating System (ONOS)!

Documentation: wiki.onosproject.org
Tutorials: tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown ONOS.

onos>
```

Figura A.1: CLI do ONOS

É necessário ativar algumas aplicações antes de trabalhar com os Switches *OpenFlows*. Use os seguintes comandos para ativá-los:

```
app activate org.onosproject.fwd
app activate org.onosproject.proxyarp
app activate org.onosproject.openflow
app activate org.onosproject.ovsdb
app activate org.onosproject.restconf
app activate org.onosproject.netcfg
feature:install onos-apps-fwd
feature:install onos-apps-proxyarp
```

Após isso abra o navegador e accesse **ip_da_máquina:8181/onos/ui/index.html**, irá aparecer a tela de login, em que usuário/senha serão onos/rocks e depois aparecerá a tela que é mostrada na Figura A.2

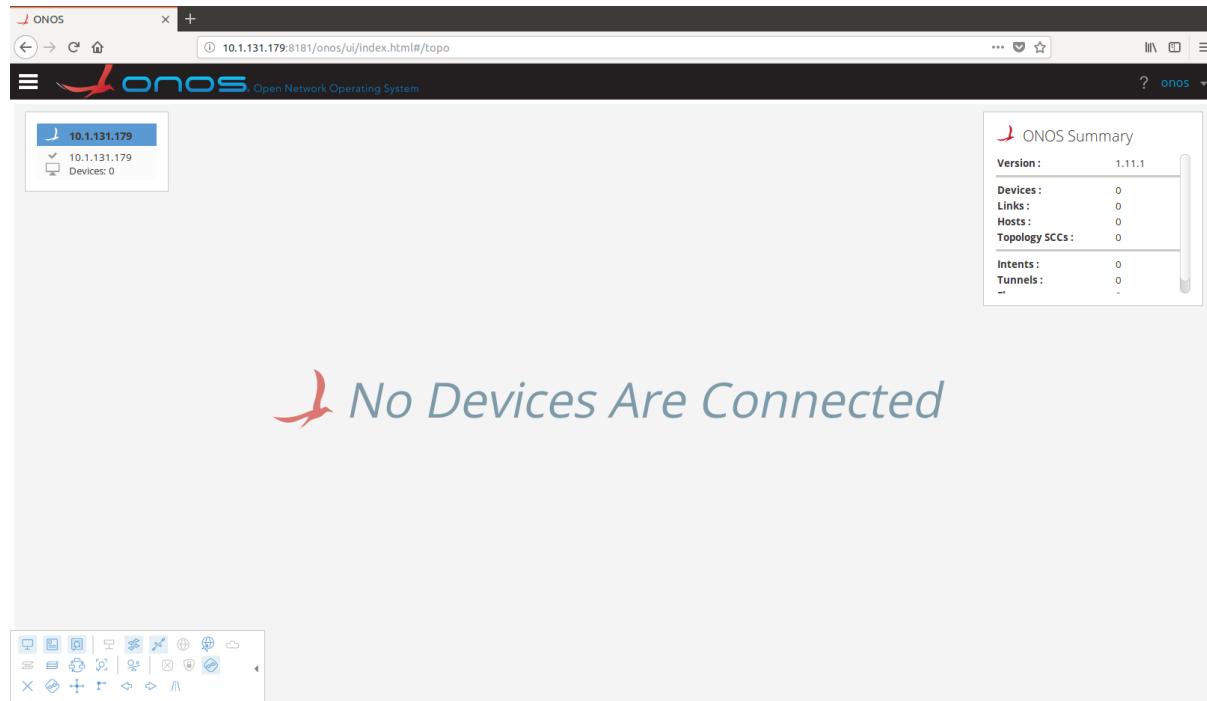


Figura A.2: Interface WEB do ONOS

Apêndice B

Implementação dos Switches *OpenFlows*

B.1 Instalação do *Open vSwitch*

```
sudo apt-get openvswitch-switch
```

B.2 Configuração do *Open vSwitch*

Criar Switch/Bridge

```
sudo ovs-vsctl add-br <nome da ponte>
```

Levantando a interface do Bridge

```
ifconfig <nome da ponte> up
```

Adicionando a porta física do PC ao switch

```
ovs-vsctl add-port <nome da ponte> <interface>
```

Retira o endereço ipv4 da interface

```
ifconfig <interface> 0
```

O bridge recebe endereço IP dinâmicamente

```
dhclient <nome da ponte>
```

No caso da rede com os Raspberrys, o endereço é setado manualmente

```
ifconfig <nome da ponte> <endereço IP> netmask <máscara de rede>
```

Torna o Switch, em OpenFlow

```
ovs-vsctl set-controller <nome da ponte> tcp:ip_controller:6633
```

Apêndice C

Configuração DHCP ONOS

C.1 Instalação e configuração do servidor DHCP

Instalação do servidor DHCP em uma máquina

```
sudo apt-get install isc-dhcp-server
```

Configurar o arquivo dhcp.conf, e comente todas as linhas que tenha alguma configuração

```
sudo nano /etc/dhcp/dhcpd.conf
```

Iniciar o servidor DHCP

```
sudo service isc-dhcp-server start
```

C.2 Ativação e configuração do ONOS para controle do serviço DHCP

Ativando a aplicação DHCP no CLI do ONOS

```
app activate org.onosproject.dhcp
```

Configure o arquivo office-dhcp.json¹

```
{
  "apps": {
    "org.onosproject.dhcp" : {
      "dhcp" : {
        "ip": "ip do servidor dhcp",
        "mac": "máscara do servidor dhcp",
        "subnet": "máscara de rede",
        "broadcast": "endereço broadcast",
      }
    }
  }
}
```

¹Está disponível em <https://goo.gl/dUu8EF>

```
"router": "roteador/gateway",
"domain": "dominio",
"ttl": "voce opta, por padrao o arquivo vem 63",
"lease": "em segundos",
"renew": "em segundos",
"rebind": "em segundos",
"delay": "2",
"timeout": "em segundos",
"startip": "ip inicial",
"endip": "ip final"
}
}
}
}
```

Upar o arquivo office-dhcp.json. lembrando que para o seguinte comando funcionar, você deve está na pasta /opt/onos/bin

```
./onos-netcfg <endereco_ip_onos> <caminho>/office-dhcp.json
```

Apenas com esses comandos, você já tem seu servidor DHCP

Apêndice D

Configuração do Nó IoT

D.1 Esquema de Pinagem

Na Figura D.1 é apresentado os pinos do Raspberry Pi 3 Model B.

Raspberry Pi 3 GPIO Header		
Pin#	NAME	Pin#
01	3.3v DC Power	DC Power 5v 02
03	GPIO02 (SDA1 , I ² C)	DC Power 5v 04
05	GPIO03 (SCL1 , I ² C)	Ground 06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14 08
09	Ground	(RXD0) GPIO15 10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18 12
13	GPIO27 (GPIO_GEN2)	Ground 14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23 16
17	3.3v DC Power	(GPIO_GEN5) GPIO24 18
19	GPIO10 (SPI_MOSI)	Ground 20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25 22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08 24
25	Ground	(SPI_CE1_N) GPIO07 26
27	ID_SD (I ² C ID EEPROM)	(I ² C ID EEPROM) ID_SC 28
29	GPIO05	Ground 30
31	GPIO06	GPIO12 32
33	GPIO13	Ground 34
35	GPIO19	GPIO16 36
37	GPIO26	GPIO20 38
39	Ground	GPIO21 40

Figura D.1: Pinos do Raspberry Pi 3 Model B e suas funções. Retirado de [Minatel 2017]

Na Figura D.2 há a configuração de pinagem e circuito usado nesse trabalho¹.

D.2 Material usado

Material usado:

- 1 Raspberry Pi 3 Model 3;
- 1 protoboard;

¹Foi usado o programa Fritzing para representar o esquema.

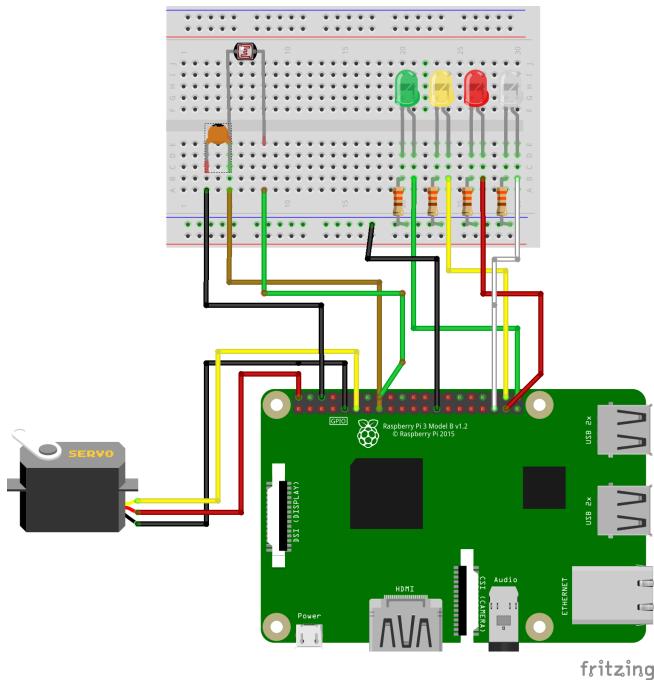


Figura D.2: Esquema de pinagem e circuito usado nesse trabalho para o N o IoT

- 4 Resistores de 330ohms;
 - 1 Capacitor² cerâmico de $1\mu F$;
 - 1 Sensor de Luminosidade LDR 5mm;
 - 4 LEDs;
 - 1 servo motor;
 - Jumpers Macho/Macho e Macho/Fêmea.

D.3 Código Embarcado da aplicação do Nó IoT

Esse código está disponível em <https://goo.gl/roCNhz>

```
# -*- coding: utf-8 -*-
#Autor: Joab de Araújo
#Código para uso em aplicação IoT no Trabalho de Conclusão de Curso -
    TCC
#Título: APLICAÇÃO DE REDES DEFINIDAS POR SOFTWARE EM INTERNET DAS
    COISAS
#Data: 25/03/2018

import RPi.GPIO as GPIO
```

²Não havia em mãos esse capacitor, então foi usado 10 capacitores de $100nF$ em paralelo, para dar o total de $1\mu F$

```
import time
import paramiko
from scp import SCPClient

#Inicia a sessao SSH
ssh = paramiko.SSHClient()
#Add o computador remoto ao arquivo Trusted do SSH
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
#Conecta ao host remoto
ssh.connect('ip_remoto', username='usuario_remoto', password='
senha_remota')

#Inicia a sessao de transferencia de Arquivo
ftp_client=ssh.open_sftp()

#Setando o modo do GPIO para BCM
GPIO.setmode(GPIO.BCM)

#Definindo variaveis para os pinos
mpin=22
tpin=23
ledVerde=21
ledAmarelo=20
ledVermelho=26
ledBranco=19
servo_pin = 17

#Definindo configuracoes para o sensor
cap=0.000001
adj=2.130620985
i=0
t=0

#Desabilitando mensagens de avisos
GPIO.setwarnings(False)

#Ajuste de valores para obter o intervalo completo do movimento do
servo
deg_0_pulse    = 0.5
deg_180_pulse = 2.5
f = 50.0
```

```
#Calculos dos parametros da largura do pulso
period = 1000/f
k      = 100/period
deg_0_duty = deg_0_pulse*k
pulse_range = deg_180_pulse - deg_0_pulse
duty_range = pulse_range * k

#Iniciar o pino GPIO do Servo Motor
GPIO.setup(servo_pin,GPIO.OUT)
pwm = GPIO.PWM(servo_pin,f)
pwm.start(0)

#Iniciar os pinos dos leds
GPIO.setup(ledVermelho, GPIO.OUT)
GPIO.setup(ledAmarelo, GPIO.OUT)
GPIO.setup(ledVerde, GPIO.OUT)
GPIO.setup(ledBranco, GPIO.OUT)

def set_angle(angle):
    duty = deg_0_duty + (angle/180.0)* duty_range
    pwm.ChangeDutyCycle(duty)

while True:
    GPIO.setup(mpins, GPIO.OUT) #Iniciar o pino 1 do sensor
    GPIO.setup(tpins, GPIO.OUT) #Iniciar o pino 2 do sensor
    GPIO.output(mpins, False)
    GPIO.output(tpins, False)
    time.sleep(0.2)
    GPIO.setup(mpins, GPIO.IN)
    time.sleep(0.2)
    GPIO.output(tpins, True)
    starttime=time.time()
    endtime=time.time()
    while (GPIO.input(mpins) == GPIO.LOW):
        endtime=time.time()
    measureresistance=endtime-starttime

    res=(measureresistance/cap)*adj
    i=i+1
    t=t+res #Limiar
    if i==10:
```

```
t=t/i
if t > 3000:
    print "Acima do limiar"
    #deixa o motor em 0 graus
    GPIO.output(ledVermelho, 0)
    GPIO.output(ledAmarelo, 0)
    GPIO.output(ledVerde, 0)
    GPIO.output(ledBranco, 0)
    set_angle(0)
if t >= 0 and t <= 3000:

    if t >= 0 and t <= 750:

        print(t)
        ftp_client.put('caminho_local_do_arquivo', 'caminho_remoto_onde_ficara_o_arquivo')
        #exemplo: ftp_client.put('/home/raspberry/Documentos/led2.txt', '/home/joab/Documentos/ledAmarelo.txt')
        GPIO.output(ledVerde, 1)
        GPIO.output(ledVermelho, 0)
        GPIO.output(ledAmarelo, 0)
        GPIO.output(ledBranco, 0)
        set_angle(180)

    if t > 750 and t <= 1500:

        print(t)
        ftp_client.put('caminho_local_do_arquivo', 'caminho_remoto_onde_ficara_o_arquivo')
        GPIO.output(ledAmarelo, 1)
        GPIO.output(ledVermelho, 0)
        GPIO.output(ledVerde, 0)
        GPIO.output(ledBranco, 0)
        set_angle(135)

    if t > 1500 and t <= 2250:

        print(t)
        ftp_client.put('caminho_local_do_arquivo', 'caminho_remoto_onde_ficara_o_arquivo')
        GPIO.output(ledVermelho, 1)
```

```
GPIO.output(ledAmarelo, 0)
GPIO.output(ledVerde, 0)
GPIO.output(ledBranco, 0)
set_angle(90)

if t > 2250 and t <= 3000:

    print(t)
    ftp_client.put('caminho_local_do_arquivo', '
caminho_remoto_onde_ficara_o_arquivo')
    GPIO.output(ledBranco, 1)
    GPIO.output(ledAmarelo, 0)
    GPIO.output(ledVerde, 0)
    GPIO.output(ledVermelho, 0)
    set_angle(45)

i=0
t=0
```

Referências Bibliográficas

[Brito 2014] BRITO, E. *Wireshark / Download / TechTudo*. 2014. <<http://www.techtudo.com.br/tudo-sobre/wireshark.html>>. (Acesso em 11/12/2017). 24

[CCNA 2013] CCNA. *Introdução a redes*. 2013. <<http://static-course-assets.s3.amazonaws.com/ITN50PT/module10/index.html#10.2.2.7>>. (Acesso em 07/03/2018). xv, 23, 30

[David 2011] DAVID, W. *Administração de Redes: Handshake ou aperto de mão*. 2011. <<http://wdredes.blogspot.com.br/2011/05/handshake-ou-aperto-de-mao.html>>. (Acesso em 07/03/2018). 20

[Flowgrammable 2012] FLOWGRAMMABLE. *SDN / OpenFlow / Message Layer / Flowgrammable*. 2012. <<http://flowgrammable.org/sdn/openflow/message-layer/>>. (Acesso on 11/12/2017). xv, 20, 21, 22

[Guedes *et al.* 2012] GUEDES, D. *et al.* Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2012*, v. 30, n. 4, p. 160–210, 2012. 1, 5, 10, 11

[Hakiri *et al.* 2015] HAKIRI, A. *et al.* Publish/subscribe-enabled software defined networking for efficient and scalable iot communications. *IEEE communications magazine*, IEEE, v. 53, n. 9, p. 48–54, 2015. xv, 1, 9

[Hamilton 2009] HAMILTON, J. *Networking: The Last Bastion of Mainframe Computing à Perspectives*. 2009. <<http://perspectives.mvdirona.com/2009/12/networking-the-last-bastion-of-mainframe-computing/>>. (Acesso em 23/11/2017). 5

[Jararweh *et al.* 2015] JARARWEH, Y. *et al.* Sdiot: a software defined based internet of things framework. *Journal of Ambient Intelligence and Humanized Computing*, Springer, v. 6, n. 4, p. 453–461, 2015. 9

[Kim e Feamster 2013] KIM, H.; FEAMSTER, N. Improving network management with software defined networking. *IEEE Communications Magazine*, IEEE, v. 51, n. 2, p. 114–119, 2013. 5

[Kreutz *et al.* 2015] KREUTZ, D. *et al.* Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, IEEE, v. 103, n. 1, p. 14–76, 2015. xv, 6

[Liberato *et al.* 2014] LIBERATO, A. *et al.* *Avaliação de Desempenho de Plataformas para Validação de Redes Definidas por Software*. [S.l.]: CSBC, 2014. 1

[Martinez-Julia e Skarmeta 2014] MARTINEZ-JULIA, P.; SKARMETA, A. F. Extending the internet of things to ipv6 with software defined networking. 2014. 9

[McKeown *et al.* 2008] MCKEOWN, N. *et al.* Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 38, n. 2, p. 69–74, 2008. 6, 10, 11

[Minatel 2017] MINATEL, P. *Os primeiros passos com Raspberry Pi 3: Controlando a GPIO por SYSFS - Blog da Usinainfo.* 2017. <<http://blog.usinainfo.com.br/os-primeiros-passos-com-raspberry-pi-3-controlando-gpio-por-sysfs/>>. (Acesso em 24/03/2018). xv, 39

[Mininet 2018] MININET. *Mininet Overview - Mininet.* 2018. <<http://mininet.org/overview/>>. (Acesso em 23/11/2017). 6

[Nunes *et al.* 2014] NUNES, B. A. A. *et al.* A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, IEEE, v. 16, n. 3, p. 1617–1634, 2014. xv, 5, 10

[ONF 2011] ONF. *Software-Defined Networking (SDN) Definition - Open Networking Foundation.* 2011. <<https://www.opennetworking.org/sdn-definition/>>. (Acesso em 04/12/2017). 5

[ONOS 2014] ONOS. *Whitepaper ONOS Final.* 2014. <<http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf>>. (Acesso em 23/11/2017). 7

[ONOS 2016] ONOS. *Agile Controller 3.0 Archives - ONOS.* 2016. <<https://onosproject.org/tag/agile-controller-3-0/>>. (Acesso em 25/03/2018). 31

[ONOS 2017] ONOS. *Wiki Home - ONOS - Wiki.* 2017. <<https://wiki.onosproject.org/display/ONOS/Wiki+Home>>. (Acesso em 23/11/2017). 7

[Pantuza 2017] PANTUZA, G. *OpenvSwitch - guia de comandos e configuração.* 2017. <<https://blog.pantuza.com/tutoriais/openvswitch-guia-de-comandos-e-configuracao>>. (Acesso em 24/11/2017). 9

[Pettit *et al.* 2010] PETTIT, J. *et al.* *Virtual switching in an era of advanced edges.* [S.l.]: Sep, 2010. 11

[Pfaff *et al.* 2009] PFAFF, B. *et al.* Extending networking into the virtualization layer. In: *Hotnets.* [S.l.: s.n.], 2009. xv, 11, 12

[Rothenberg *et al.* 2010] ROTHENBERG, C. E. *et al.* Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. *Cad. CPqD Tecnologia, Campinas*, v. 7, n. 1, p. 65–76, 2010. 5

[Subramanian e Voruganti 2016] SUBRAMANIAN, S.; VORUGANTI, S. *Software Defined Networking (SDN) with OpenStack.* [S.l.]: Packt Publishing - ebooks Account, 2016. ISBN 178646599X. xv, 7, 11