Universidade Federal do Estado do Rio de Janeiro – UNIRIO

Centro de Ciências Exatas e Tecnologia

Programa de Pós-Graduação em Informática

Um Framework para Experimentos Realísticos em Redes Sem Fio Definidas por Software

Roberto Gerpe Arman Mendes Barros

Orientador:

Prof. Dr. Carlos Alberto Vieira Campos

RIO DE JANEIRO, RJ, BRASIL

OUTUBRO DE 2016

Um Framework para Experimentos Realísticos em Redes Sem Fio Definidas por Software

Roberto Gerpe Arman Mendes Barros

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO), APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Carlos Alberto Vieira Campos, D. Sc. - UNIRIO

Sidney Cunha de Lucena, D. Sc. - UNIRIO

Bruno Astuto Arouche Nunes , Ph.D. - GE GLOBAL RESEARCH

RIO DE JANEIRO, RJ, BRASIL
OUTUBRO DE 2016

Barros, Roberto Gerpe Arman Mendes.
B277 Um *framework* para experimentos realísticos em redes sem fio definidas por *software* / Roberto Gerpe Arman Mendes Barros, 2016.
98 f. : il. (algumas color.) ; 30 cm

Orientador: Carlos Alberto Vieira Campos.
Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2016.

1. Redes de computadores. 2. Sistemas de comunicação sem fio. 3. Redes Definidas por Software. 4. Framework (Arquivo de computador). I. Campos, Carlos Alberto Vieira. II. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. III. Título.

CDD – 004.6

Dedico este trabalho para a minha filha Amanda e meus pais. Com o seu amor e incentivo sempre me estimulou a seguir meus projetos de vida. De muitas formas me incentivaram e me ajudaram para que fosse possível este trabalho.

Agradecimentos

Agradeço a todas as pessoas do meu convívio que acreditaram e contribuíram, mesmo que indiretamente, para a conclusão deste curso.

Ao meu orientador pelo desprendimento, paciência e dedicação que me motiva a sempre me aprimorar na pesquisa científica e me ensinar o verdadeiro significado de fazer ciência. Agradeço muito pelo empenho, paciência e inclusive pela cobrança de forma firme, cujo único propósito era o meu sucesso pessoal. Minha enorme gratidão, sem a sua orientação, não teria chegado até aqui. Acreditou em meu potencial, tornando-se mais que um orientador, um amigo verdadeiro ao longo de toda a jornada, uma referência a ser seguida como educador neste país.

Resumo

As redes sem fio heterogêneas são redes que utilizam concorrentemente múltiplas tecnologias sem fio. Essas redes possuem alta complexidade de operação e podem possuir dinamicidade em sua infraestrutura, como por exemplo, frequentes associações e dissociações dos dispositivos na rede, motivadas pela diversidade de características e restrições de cada uma de suas tecnologias. Tais problemas perpassam a gestão, a operação, a manutenção e a reconfiguração ao tornar a topologia complexa ou ocorrer um aumento de escala no número de nós sem fio. As Redes Definidas por Software (Software-Defined Networks SDN) possibilitam a separação do plano de dados e do plano de controle, o que muda a concepção, construção e os testes destas redes. O suporte em SDN para redes sem fio ainda está incipiente em comparação com as redes cabeadas. A dissociação do plano de controle é vantajosa ao permitir uma configuração única e centralizada e agilizar a automatização da tomada de decisão para os dispositivos de rede que antes eram tratados individualmente. Em contrapartida, podemos destacar desafios de consistência quando ocorrem mudanças frequentes e de escalabilidade quanto ao número de nós e fluxos.

Dentro deste contexto, esta dissertação propõe um *framework* realístico que possibilite aos pesquisadores de rede sem fio desenvolverem suas propostas, experimentos e análises sob uma perspectiva SDN, com destaque para a mobilidade dos usuários. Este *framework* foi implementado como uma extensão do protocolo Openflow, através de mensagens do tipo "Experimenter", o que permite o recebimento pela controladora de informações dos dispositivos sem fio e também o envio de comandos para estes. A implementação do FRESDWN é dividida em três componentes básicos: a controladora externa Openflow (RYU), o switch Openflow customizado (ofsoftswitch13) e o ambiente integrado MININET versão 2.2.1 com NS-3 versão 3.22. Foi realizada uma emulação baseada em simulação ao utilizar o MININET para se comunicar com o controladora RYU e o simulador de redes NS-3 para representar a mobilidade dos nós sem fio e também os componentes da camada Física e de Enlace. Através dos estudos de caso com baixa e com alta mobilidade, implementamos um mecanismo na controladora de desassociação antecipada, com base nas informações recebidas dos pontos de acesso. Foram obtidos vários resultados em relação ao número de desassociações dos nós móveis em relação ao tempo e também à proporção das mensagens Experimenter em relação às demais mensagens do plano de controle. Por meio dos resultados obtidos, foi evidenciada a viabilidade e a efetividade do *framework*, e também mostrou que a abordagem de redes sem fio definidas por software é factível e permite atuar sobre os diversos problemas de gerenciamento de redes sem fio.

Palavras-chave: Redes de Computadores, Redes sem fio, Redes sem fio heterogêneas, SDN, SDWN,*Framework*.

Abstract

The heterogeneous wireless networks are networks that use multiple wireless technologies concurrently. These networks are highly complex operation and may have dynamism in its infrastructure, such as frequent associations and dissociations of the devices on the network, driven by the diversity of characteristics and constraints of each of its technologies. These problems pervade the management, operation, maintenance and reconfiguration to make complex topology or occur scalability in the number of wireless nodes. The Software-Defined Networks (SDN) enables the separation of the data plane and control plane which changes the design, construction and testing of these networks. The support for SDN for wireless networks is still in its infancy compared to wired networks. The control plane decoupling is advantageous to provide a single, centralized configuration and streamline the automation of decision making to the network devices that were treated individually. On the other hand, we can highlight consistency challenges when there are frequent changes, and scalability as the number of nodes and flows.

Within this context, this work proposes a realistic framework that enables the wireless network researchers to develop their proposals, experiments and analysis under SDN perspective, with emphasis on the mobility of users. This framework has been implemented as an extension of the OpenFlow protocol, through messages "Experimenter" which allows the receipt by the controller the information from wireless devices and also sending commands to them. The implementation of FRESDWN is divided into three basic components: the external controller OpenFlow (RYU), the custom OpenFlow switch (ofsoftswitch13) and the built environment Mininet version 2.2.1 with NS-3 version 3.22. It was done a simulation-based emulation using the Mininet emulator that communicated with the controller RYU and NS-3 network simulator to represent mobility of wireless nodes and also the components of the physical layer and link was performed. Through case studies with low and high mobility, we implemented a mechanism of controlling early disassociation, based on information received from the access points. We obtained several results in the number of disassociation of mobile nodes over time and also the proportion of Experimenter messages in relation to other control plan messages. Through the results obtained, It was shown the viability and effectiveness of the framework, and also showed that the approach of software defined wireless networks is feasible and allows to work on various wireless networks management issues.

Keywords: Computer networks, wireless networks, heterogeneous wireless networks, SDN, SDWN, Framework.

Sumário

1 INTRODUÇÃO . . . 1
1.1 Motivação . . . 2
1.2 Problema . . . 4
1.3 Proposta de Solução . . . 5
1.4 Estrutura da Dissertação . . . 7

2 REVISÃO DA LITERATURA . . . 9
2.1 Redes Definidas por Software . . . 9
2.1.1 ForCES . . . 15
2.1.2 SANE e Ethane . . . 18
2.1.3 Openflow . . . 19
2.1.3.1 Características Básicas . . . 20
2.1.3.2 Controladora Openflow . . . 20
2.1.3.3 *switch* Openflow . . . 21
2.1.3.4 Mensagens Openflow . . . 21
2.2 Redes Sem Fio Definidas Por Software . . . 24
2.2.1 ODIN e OpenSDWN . . . 25
2.2.2 wmSDN . . . 26
2.2.3 MININET-WIFI . . . 27
2.2.4 OpenNet . . . 28
2.2.5 Aetherflow . . . 29
2.2.6 Consideracoes Finais . . . 30

3 PROPOSTA DE SOLUÇÃO . . . 32
3.1 Definição do FRESDWN . . . 32
3.2 Implementação do FRESDWN . . . 35

4 AVALIAÇÃO DE DESEMPENHO DE UM ESTUDO DE CASO . . . 45
4.1 Contextualização e objetivos do estudo de caso do FRESDWN . . . 45
4.2 Descrição do Cenário . . . 46
4.3 Configurações e parâmetros . . . 49
4.4 Análise do Cenário com baixa mobilidade . . . 50
4.4.1 Análise do Cenário com alta mobilidade de nós . . . 54

5 CONCLUSÃO . . . 59
5.1 Discussões Finais . . . 59

5.2 **Contribuições** **59**
5.3 **Trabalhos Futuros** **60**

REFERÊNCIAS **62**

ANEXOS **69**

ANEXO A – OFSOFTSWITCH13 . . . . . **70**
A.1 **Makefile.in** **70**
A.2 **Makefile** **70**
A.3 **lib** **76**
A.3.1 lib/automake.mk 76
A.4 **ofl-exp** **76**
A.4.1 ofl-exp/ofl-exp.c 76
A.4.2 ofl-exp/ofl-exp-fresdwn.h 76
A.4.3 oflib-exp/.deps/ofl-exp.Po 77
A.4.4 ofl-exp/ofl-exp-fresdwn.c 78
A.4.5 oflib-exp/automake.mk 82
A.5 **udatapath** **82**
A.5.1 udatapath/dp$_f resdwn.h$ 82
A.5.2 udatapath/dp$_f resdwn.c$ 82
A.5.3 udatapath/automake.mk 83
A.5.4 udatapath/dp$_e xp.c$ 83
A.5.5 udatapath/.deps/udatapath$_o f datapath - dp_e xp.Po$ 83
A.6 **include/openflow** **84**
A.6.1 include/openflow/fresdwn-ext.h 84
A.6.2 include/openflow/automake.mk 87

ANEXO B – CONTROLADORA RYU **88**
B.1 **run.py** **88**
B.2 **main$_a pp.py$** **88**

ANEXO C – AMBIENTE INTEGRADO MININET/NS-3 . . . . . **93**
C.1 **wifi$_m obility.py$** **93**

Lista de ilustrações

Figura 1 – A arquitetura SDN adaptada de (HALEPLIDIS et al., 2015a) 10
Figura 2 – A arquitetura ForCES adaptado de HALEPLIDIS et al., 2015b 16
Figura 3 – A arquitetura de abstração do FE e entrada e saída do LFB do ForCES adaptada de HALEPLIDIS et al., 2015b 17
Figura 4 – Ilustra a comunicação entre os nós sem fio móvel e a controladora . . . 22
Figura 5 – Ilustra a topologia e comunicação da arquitetura ODINSURESH et al., 2012 25
Figura 6 – Ilustra a troca de mensagens Experimenter sem confirmação entre a controladora e o nó sem fio móvel (*switch* Openflow), quando a mensagem do FRESDWN foi iniciada pela controladora 33
Figura 7 – Ilustra a troca de mensagens Experimenter sem confirmação entre a controladora e o nó sem fio móvel, quando a mensagem do FRESDWN foi iniciada pelo nó movel (*switch* openflow) 33
Figura 8 – Ilustra a troca de mensagens Experimenter com confirmação entre a controladora e o nó sem fio móvel (*switch* openflow), quando a mensagem do FRESDWN foi iniciada pela controladora 34
Figura 9 – Ilustra a troca de mensagens Experimenter com confirmação entre a controladora e o nós sem fio móvel, quando a mensagem do FRESDWN foi iniciada pelo nó móvel (*switch* openlflow) 34
Figura 10 – Ilustra a topologia e comunicação entre os nós sem fio móvel e a controladora 35
Figura 11 – Detalhamento da Topologia entre Controladora externa, MININET e NS-3 36
Figura 12 – Detalhamento da Ponte TAP da Topologia entre MININET e NS-3 . . 37
Figura 13 – as especificação da mensagem vendor do Openflow 1.0, adaptada do flowgrammable. 38
Figura 14 – segundo as especificação do Openflow 1.1 em diante, adaptada do flowgrammable. 39
Figura 15 – Disposição dos pontos de acesso e definição da área delimitada de movimentação dos nós nos cenários 47
Figura 16 – Disposição dos pontos de acesso e raio de propagação com potência de sinal de $-72dBm$ nos cenários, ilustrando a área aonde aconteceria a desassociação 48
Figura 17 – Disposição do primeiro cenário com baixa mobilidade, com troca de dados entre dois nós no ambiente emulado, sendo H1 um nó estático e H2 um nó com mobilidade 51

Figura 18 – Disposição do primeiro cenário com baixa mobilidade, com troca de dados entre dois nós no ambiente emulado, sendo H1 um nó estático e H2 um nó com mobilidade 52
Figura 19 – Resultado dos tempos totais das desassociações do nó H2 no cenário com baixa mobilidade 53
Figura 20 – Percentual de mensagens Experimenter em relação às demais outras mensagens Openflow trocadas pela controladora com os pontos de acesso no cenário com baixa mobilidade 54
Figura 21 – Disposição do cenário com alta mobilidade com troca de dados entre os três nós móveis no ambiente emulado. 54
Figura 22 – Resultado dos tempos totais das desassociações do nó H2 no cenário com alta mobilidade 55
Figura 23 – Taxa de utilização da mensagem Experimenter em relação as demais outras mensagens Openflow trocadas pela controladora com os pontos de acesso no cenário com alta mobilidade 56

Lista de tabelas

Tabela 1 – Comparativo entre SDN e redes tradicionais, baseado em (XIA et al., 2015a) 9
Tabela 2 – Comparativo entre os tipos das mensagens em cabeçalho Openflow (até o suportado pelo Openflow 1.0) 23
Tabela 3 – Comparativo entre os tipos das mensagens em cabeçalho Openflow (após o suportado pelo Openflow 1.0) 24
Tabela 4 – Parâmetros utilizados nos estudos de casos 50
Tabela 5 – Valores absolutos das mensagens Openflow em relação ao Número de desassociações 53
Tabela 6 – Valores absolutos das mensagens Openflow em relação ao Número de desassociações 57

Lista de códigos fontes

1 Alterações do Makefile.in do Ofsoftswitch13 70
2 Alterações do Makefile do Ofsoftswitch13 76
3 Alterações do lib/automake.mk 76
4 Acrescentar em ofl-exp.c 76
5 ofl-exp-fresdwn.h 77
6 Alterações do oflib-exp/.deps/ofl-exp.Po 78
7 ofl-exp/ofl-exp-fresdwn.c 81
8 Alterações do oflib-exp/automake.mk 82
9 $dp_fresdwn.h$ 82
10 udatapath/$dp_fresdwn.c$ 83
11 Alterações do udatapath/automake.mk 83
12 Alterações do udatapath/$dp_exp.c$ 83
13 Alterações do udatapath/.deps/udatapath$_of datapath - dp_exp.Po$ 83
14 include/openflow/fresdwn-ext.h 87
15 Alterações do include/openflow/automake.mk 87
16 run.py 88
17 main$_app.py$ 92
18 wifi$_mobility.py$ 98

Lista de símbolos

AAA	Authentication, authorization, and accounting
ACID	Atomicity, consistency, isolation, and durability
API	Application Programming Interface
APs	Access Points ou Pontos de acesso sem fio
BSS	Basic Service Set
BSSID	Basic Service Set Identification
CapEx	Capital Expenditure
CBR	Constant Bit Rate
CDMA	Code division multiple access
CE	Control element
CES	Customer edge switching
CP	Control plane
DP	Data plane
ESS	Extended Service Set
ESSID	Extended Service Set Identification
FE	Forwarding element
GPL	General public license
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPC	Inter-Process Communication
ISM	Industrial, Scientific and Medical radio bands

ITU	International Telecommunication Union
L1	Layer 1
L2	Layer 2
L3	Layer 3
L4	Layer 4
L7	Layer 7
LAN	Local area network
MAC	Media access control
MBR	Maximum bit rate
NetConf	Network configuration é um protocolo para configuração de elementos de redes
NGN	Nextgeneration network
OF	OpenFlow
ONF	Open Networking Foundation
REST	Representational State Transfer
RF	radiofrequência
RFC	Request for Comments
RPC	Remote Procedure Call
RSS	Received Signal Strength
RTP	RealTime Transport Protocol
RTS	sinal de Request To Send
RTT	Round-Trip Time
SBI	Southbound interface
SDMN	Softwaredefined mobile networks
SDN	Software-Defined Network
SDNRG	SoftwareDefined Networking Research Group

SDR	Softwaredefined radio
SDS	Softwaredefined security
SDWN	Software-Defined Wireless Network
SNIR	Signal to Noise plus Interference Ratio
SNMP	Simple Network Management Protocol
SNR	Signal-to-Noise Ratio
SSID	Service-Set IDentification
Wi-Fi	marca registrada da Wi-Fi Alliance utilizada por produtos certificados que pertencem à classe de dispositivos de rede local sem fio baseados no padrão IEEE 802.11
WLAN	Wireless Local Area Network
WMN	Wireless Mesh Networks
WPAN	Wireless personal area network
WWAN	Wireless wide area network

1 Introdução

Tradicionalmente as redes de computadores costumam utilizar dispositivos de encaminhamento com *hardware* dedicado e de propósito específico, o que permite uma comunicação de dados com alto desempenho, mas, em contrapartida, impõe, além de grandes dificuldades, também altos custos, tanto para os usuários quanto para as empresas (COMMITTEE et al., 2012). Estes equipamentos de *hardware* dedicados são utilizados na montagem de redes para aplicações críticas, normalmente em topologias complexas e, na grande maioria das vezes, com requisitos conflitantes. A configuração e instalação pelos administradores de rede destes equipamentos requerem conhecimentos técnicos bem especializados e, para desempenharem tais tarefas, devem ser altamente treinados. Não obstante este treinamento, costuma-se obter desempenho apenas moderado, em função da complexidade das tarefas (SEZER et al., 2013).

Além disto, estes administradores devem lidar com os custos operacionais envolvidos no provisionamento e no gerenciamento de suas redes. Na grande maioria dos casos possuem parques tecnológicos heterogêneos de tecnologias e também de múltiplos fornecedores, o que dificulta a constante busca de reduzir o valor das despesas operacionais - OPEX (*Operational Expenditures*)(CHATER; CHAFNAJI, 2014). A redução de recursos humanos especializados e o aumento dos custos crescentes fomentam um interesse em soluções que possam unificar o gerenciamento de rede e seu provisionamento em vários domínios (Han; Ferreira; Costeira, 2016).

A comunicação móvel é atualmente o principal meio que os consumidores contratam dos MNOs (*mobile network operators*) e vem aumentando em escala mundial. No final de 2015, já ultrapassou os 7 bilhões de dispositivos móveis e 3.2 bilhões de usuários de Internet, sendo 2 bilhões em países em desenvolvimento (ITU, 2015). Dos dispositivos móveis, destacam-se os *smartphones*, pois se tornaram o padrão para o uso da rede e as suas vendas ultrapassaram 1,2 bilhões de unidades, de um total de 1,8 bilhões de celulares vendidos somente no ano de 2014 (GARTNER, 2015). O vertiginoso crescimento e desenvolvimento dos *smartphones* criaram plataformas tecnológicas multifacetadas que representam a convergência acelerada de telefonia móvel, serviços de Internet e computação pessoal (CAMPBELL-KELLY et al., 2015). O uso das aplicações nos *smartphones* variam, desde aplicações mais comuns à maioria dos usuários como navegação *web*, *e-mail*, mensagens instantâneas, vídeo sob demanda, vídeo-chamadas, jogos *online*, *homebanking* e redes sociais, até aplicações mais específicas para um grupo específico de usuários como músicos, engenheiros, dentre outros.

1.1 Motivação

A principal mudança de paradigma nas tecnologias de comunicação foi a dos meios estritamente guiados para a utilização, cada vez mais frequente, dos meios sem fio (GARTNER, 2015). Esta é uma das razões para a densificação do uso do espectro eletromagnético de transmissão sem fio, motivado pela necessidade de mobilidade e de acesso ubíquo, o que apresenta um grande obstáculo a ser superado pelas Redes Sem Fio. Tornar os serviços sempre disponíveis é um dos problemas mais desafiadores em um ambiente de redes heterogêneas, exigindo melhorias na forma como o dispositivo móvel classifica e escolhe o seu ponto de acesso (MONTEIRO et al.,).

As Redes Sem Fio estão presentes em praticamente todos os ambientes, tanto nos corporativos, quanto nos residenciais, tanto para estender o alcance das redes locais, como para prover acesso à Internet. Um dos principais motivadores para essa crescente adoção dessas redes é a possibilidade de atender à demanda dos usuários móveis. Portanto, tomar a melhor decisão de *handover*[1], baseada na escolha do processo de seleção de rede, é um fator importante na continuidade de forma transparente da transmissão de dados pelo terminal móvel (BOULHOSA et al., 2011). Para minimizar os problemas de interferências e aumentar a segurança e/ou velocidade, costuma-se reduzir a área de cobertura do sinal de rede sem fio de um ponto de acesso.

O meio sem fio apresenta uma série de novos problemas relativos a seus atributos de mobilidade, multi-percursos, controle de sinal, diagrama de propagação do sinal, segurança no meio compartilhado, dentre outros. Isto faz com que a cada novo dispositivo que entre na rede exija um ajuste, na maioria das vezes manual, na gestão do dispositivo. Tais manutenções ou correções de problemas, além de se tornarem um trabalho repetitivo e tedioso, são passíveis de erros, porque podem exigir, no pior cenário, a reconfiguração individual de cada um dos muitos dispositivos de rede. Diversas soluções proprietárias procuram resolver os problemas decorrentes do adensamento dos pontos de acesso, como maior interferência entre os dispositivos ou a maior quantidade de troca das estações entre os pontos de acesso.

As abstrações na modelagem e recursos em meios não guiados são mais complexas do que nos meios guiados. Tal complexidade se deve à natureza intrínseca das comunicações sem fio como, por exemplo, a variação estocástica da qualidade do canal de transmissão (LIANG; YU, 2015).

O gerenciamento dos dispositivos heterogêneos de rede sem fio costumam sofrer pela falta de interoperabilidade entre as soluções dos diversos fabricantes, o que se torna um processo lento e cansativo para os administradores de rede, além do fato de, algumas

[1] *handover* também conhecido como *handoff* é a transição de um nó móvel de uma célula para outra célula de forma transparente ao utilizador em redes sem fio.

vezes, não conseguirem viabilizar a implantação de uma solução que realmente atenda às suas necessidades.

O paradigma de pesquisa e implementação na área de redes tem sido intensamente alterado devido ao surgimento das Redes Definidas por *software* (*software*-Defined Networking - SDN)NIEPHAUS et al., 2015. SDN é uma abordagem para redes de computadores que permite gerenciá-la por meio de uma abstração, na qual há o desacoplamento da parcela que, efetivamente, toma decisões sobre o fluxo de dados (plano de controle) e da parcela que é responsável pelo encaminhamento dos dados (plano de dados). SDN é uma sigla cada vez mais comum no mundo de Tecnologia da Informação e Comunicação. Atualmento SDN é um tópico quente dentro da comunidade científica, sendo que praticamente qualquer evento na área de redes de computadores possui direta ou indiretamente um tema relacionado ao mesmo. Isto pode ser constatado por investimentos em pesquisa na ordem de milhões de dólares, bem como pela premiação da National Science Foundation, órgão de fomento à pesquisa nos Estados Unidos da América (NSF.GOV, 2015).

Dentre as diversas soluções SDNs, destacam-se as que utilizam o protocolo Openflow (MCKEOWN et al., 2008), como seu protocolo de troca de informações do plano de controle e de definição do comportamento do plano de dados. A ONF (*Open Networking Foundation*) define que, na arquitetura SDN, os planos de controle e de dados são dissociados, a inteligência de rede e seus estados são logicamente centralizados e a infraestrutura de rede é exposta para as aplicações (COMMITTEE et al., 2012). Há uma previsão de crescimento de 35 bilhões de dólares até 2018 no mercado de SDN (SDXCENTRAL, 2015). O protocolo Openflow não atua sobre os aspectos de configuração e gerenciamento de redes, e despreza as particularidades das Redes Sem Fio. Para melhorar o desempenho dessas redes são necessárias novas abordagens, que os pesquisadores devem propor, testar e validar. Infelizmente, as funções, atualmente disponibilizadas em controladoras WLAN (*Wireless Local Area Networks*), são fechadas e proprietárias, sendo insuficientes para permitir a inovação. Uma solução para contornar essa limitação é a utilização de API (*Application Programming Interfaces*) abertas como o Openflow, que permitam o desenvolvimento de aplicações e algoritmos de controle sensíveis ao contexto.

O protocolo Openflow foi projetado para atender à evolução e extensão dos elementos do SDN, sendo a sua API baseada em um grande número de casos de uso não correlacionados de SDN e diretrizes clássicas de desenvolvimento de APIs com o objetivo de atender a uma ampla gama de dispositivos de rede, tanto em *hardware* quanto em *software*. Deve-se garantir flexibilidade e extensibilidade, pois cada característica deve ser definida do modo mais amplo possível, sem limitações arbitrárias, para poder ser aplicada a um amplo espectro de implementações e casos de uso. As extensões *Experimenter* são o caminho para que a inovação ocorra, ao permitir um mecanismo de extensão que habilita o pesquisador a estender alguma estrutura de um objeto especificado (mensagens, ações,

erros)(OPENNETWORKING.ORG, 2016).

1.2 Problema

A inovação na arquitetura de redes tem sido motivada também pela virtualização, o que permite que as funções de redes tenham sido movidas para a nuvem com o NFV (*Network Function Virtualization*), além do controle centralizado das redes SDN e, através destes dois princípios, é possível atender a essa demanda crescente, além de permitir inovação e redução custos dos provedores de serviços móveis (NAUDTS et al., 2016).

Segundo (XIA et al., 2015b), SDN deve contemplar todas as possibilidades de meios de transmissão, incluindo par metálico, ótico e sem fio, a fim de atender a essa ubiquidade. Esses diferentes meios de transmissão, cada qual com as suas características particulares, não só em termos de tecnologias de configuração como de gerência, devem ser analisadas. Para tanto, o SDN deve ser capaz de se integrar com essas redes, em especial as óticas e as sem fio. A integração dessas tecnologias propicia, à controladora SDN, uma grande oportunidade de possuir um controle mais amplo e eficiente sobre o comportamento da rede, destacando-se o encaminhamento de pacotes, modo de operação ou canal sem fio.

Devido à diversidade de tecnologias de transmissão e à crescente utilização das Redes Sem Fio, principalmente considerando os dispositivos móveis, podem ocorrer diversos problemas. Podemos destacar problemas com os limites dos recursos de rádio transmissão, autonomia de bateria, comunicações intermitentes, dentre outros (MENDONCA et al., 2013; LEI et al., 2013).

Os dispositivos sem fio estão cada vez mais utilizando múltiplas tecnologias de rádio transmissão, o que caracteriza essas Redes Heterogêneas (TANG; LIAO, 2014). As Redes Sem Fio Heterogêneas possuem os seguintes desafios em SDN: otimização de recursos em ambientes dinâmicos, a granularidade dos controles e das políticas de gerenciamento (OPENNETWORKING.ORG, 2015c). Apesar dos avanços nas pesquisas, até onde se pode constatar, ainda não existe um *framework* que permita a inovação dessa diversidade, em um ambiente replicável realístico, que explore o potencial do protocolo Openflow aplicado a Redes Sem Fio.

Além de possuirmos atualmente muitos padrões de Redes Sem Fio, continuam surgindo novos padrões de comunicação que possuem características próprias de raio de atuação, comprimento de onda, número de dispositivos suportado, dentre outras características. Tudo isso para atender a essa demanda reprimida por maior cobertura e melhor vazão de dados. Portanto, a configuração eficiente dos parâmetros dos APs (*access points*) pode melhorar significativamente a capacidade e desempenho das redes WLAN (*Wireless Local Area Network*. Para ilustrar esses novos padrões, podemos referenciar o 802.11ac

(IEEE, 2013), que é um padrão de WiFi que, espera-se, atenda a algumas dessas demandas de Redes Sem Fio ao suportar taxas de transmissão de dados de até 6.93Gbps, com MIMO multi-usuário, agregação de quadros e transmissões de banda larga. Um trabalho que propõe resolver o problema dessas tecnologias de WLAN adota canais na faixa de 5MHz a 160MHz, o que transformaria esse problema de canais heterogêneos em um problema FSS (*fine-grained spectrum sharing*), e viabilizaria uma solução justa e eficiente (HAN; ZHANG; SHIN, 2016).

As abordagens atuais, para gerenciamento da rede sem fio em APs *standalone* de baixo custo, não são escaláveis por consistirem em configurações de parâmetros em cada um dos dispositivos, além do fato de possuírem algoritmos simplificados para escolha do canal e da potência de transmissão. Para atender à escalabilidade, podem utilizar soluções corporativas com a aquisição de equipamentos proprietários caros que permitam uma gestão centralizada, mas não permitem orquestrar um ambiente heterogêneo (D-LINK, 2015b; HP, 2015; CISCO, 2015b). Estes algoritmos são genéricos e mantidos pelo fabricante, e, portanto, não costumam atender a demandas específicas dos administradores de rede, pois dependem dos fabricantes acreditarem que alguma alteração seja relevante para realização de *patches* ou atualizações. Existem alguns fornecedores de soluções proprietárias que oferecem a gestão, sem a necessidade de adquirir um equipamento para tal fim (CISCO, 2015a; D-LINK, 2015a). Um serviço para as universidades federais, que é baseado em *software*, permite gerir APs de baixo custo, oferecendo à comunidade acesso sem fio seguro para a comunidade internacional de educação e pesquisa (RNP, 2015). Uma solução de código livre é o OpenWISP (OPENWISP.ORG, 2015) que permite gerenciar, de forma centralizada, um grande número de pontos de acesso de uma maneira simples, desde que o AP possua uma versão customizada do *firmware* OpenWrt (OPENWRT.ORG, 2015).

A escolha da melhor rede de acesso, em um ambiente de tecnologias heterogêneas, tem caracterizado a fase de seleção de rede dentro do gerenciamento de *handover* como um grande obstáculo a ser superado pelas NGMN (*Next-Generation Mobile Networks*), visto que irá afetar diretamente alguns aspectos inerentes à QoS do ponto de acesso selecionado, visando dar as melhores condições de conexão às transmissões do usuário. Em suma, o problema que nos motivou decorre da percepção de que os modelos atuais de gerenciamento de Redes Sem Fio não conseguem atender às demandas dos administradores e dos usuários destas redes heterogêneas.

1.3 Proposta de Solução

Nesta dissertação, propomos um *framework* para uma arquitetura SDN com Redes Sem Fio, denominado *Framework for Realistic Experiments using software Defined Wireless Networks* - FRESDWN, que permita que sejam desenvolvidos *softwares* de con-

trole customizados, com informações complementares dos dispositivos, através de uma extensão do protocolo Openflow. O FRESDWN adequa as funcionalidades do plano de controle e de dados entre os clientes sem fio, pontos de acesso sem fio e o controlador SDN, sem, obrigatoriamente, efetuar alteração dos clientes sem fio. Este controlador pode ser centralizado ou distribuído, mas deve sempre manter uma visão global da rede.

O FRESDWN permite que os controladores e os dispositivos de rede sem fio possam interagir, trocando informações desse ambiente sem fio que não são trocadas pelo protocolo Openflow, em suas versões disponíveis até a presente data [OPENNETWORKING.ORG, 2009; OPENNETWORKING.ORG, 2011a; OPENNETWORKING.ORG, 2011b; OPENNETWORKING.ORG, 2012a; OPENNETWORKING.ORG, 2012b; OPENNETWORKING.ORG, 2013a; OPENNETWORKING.ORG, 2013b; OPENNETWORKING.ORG, 2014a; OPENNETWORKING.ORG, 2015a; OPENNETWORKING.ORG, 2013c; OPENNETWORKING.ORG, 2014b; OPENNETWORKING.ORG, 2015b]. Ao utilizar a mensagem *Experimenter*, o FRESDWN provê meios, aos pesquisadores, de validar suas propostas e submetê-las para que possam ser incorporadas em futuras versões do protocolo Openflow. O *FRESDWN* também pode beneficiar desenvolvedores de aplicações de redes e fabricantes. A coexistência de redes de acesso heterogêneas, na próxima geração de Redes Sem Fio, traz consigo uma diversidade de parâmetros de rede, que influenciarão diretamente nos quesitos da aplicação do usuário. Para ilustrar, quando o usuário se desloca de uma rede sem fio para outra, podendo possuir tecnologias diversas (Wi-Fi para 4G, por exemplo), é bom que, antes de executar o *handover*, um algoritmo avalie, dentre a lista de redes de acesso, utilizando múltiplas métricas, qual melhor atenderá a necessidade da aplicação de usuário, seja ela de dados, voz ou imagem.

O controlador pode gerenciar o dispositivo sem fio de qualquer tecnologia que receba uma versão customizada do *switch* Openflow. Esta versão é necessária para que se possa tratar as mensagens do tipo *Experimenter*, utilizadas pelo FRESDWN para a troca de informações com a controladora. Isto permite, por exemplo, que a controladora modifique os parâmetros dos enlaces sem fio, permitindo gerenciar o *handoff* dos dispositivos, além de interferir no processo de associação e desassociação das estações sem fio no ambiente heterogêneo.

Foi criado um ambiente emulado com pontos de acesso FRESDWN (IEEE 802.11) executando uma versão customizada do ofsoftswitch (CPQD, 2015), *switch* Openflow, implementado a nível de usuário, e do RYU (SÁNCHEZ, 2015), uma controladora centralizada, projetada para aumentar a agilidade da rede, tornando mais fácil de gerir e adaptar o tráfego de dados, que solicita e envia as mensagens *Experimenter*, conforme disponibilizado pelo FRESDWN, sendo seu funcionamento e desempenho avaliado em estudos de caso envolvendo o controle de associação cliente e identificação de problemas na interface sem fio dos APs.

Com o FRESDWN é possível desenvolver programas na controladora interagindo no plano de controle e, com isto, lidar com questões como:

- Permitir coordenação entre os APs;
- Permitir um melhor controle do *handoff* dos clientes facilitando o processo de mobilidade;
- Implementar controle adaptativo para Redes Sem Fio;
- O controle do processo de associação de estações sem fio aos APs;
- Adicionar mecanismos de configuração do plano de dados, tais como canal, potência e fluxos;
- Permitir o uso de algoritmos de controle adaptados às necessidades de cada operador;
- O controle do fluxo de quadros ARP para a rede sem fio, aumentando a disponibilidade de tempo de transmissão, em função da filtragem de tráfego desnecessário;
- A identificação de interfaces sem fio que não estejam transmitindo, utilizando recursos de varredura do protocolo 802.11;
- Reduzir os custos de capital, ao permitir que o controlador gerencie redes com equipamentos de diversos fabricantes que estejam habilitados a tratar as mensagens *Experimenter*.

Através de um protótipo, implementado como prova de conceito, foram obtidos resultados que nos mostram que a abordagem SDN é viável para Redes Sem Fio. Entretanto, devido à limitação de recursos, nossos experimentos não foram realizados em ambientes de alta concentração de usuários e pontos de acesso. Também não foram avaliadas situações de alta mobilidade de tráfego, que demandassem respostas rápidas do controlador, de forma que a latência gerada por este processo de decisão pudesse ser avaliada.

1.4 Estrutura da Dissertação

Esta dissertação está organizada conforme descrita a seguir. No Capítulo 2 será abordada a fundamentação teórica dos temas abordados, percorrendo os fundamentos de uma rede sem fio e detalhando suas arquiteturas, seguindo para as suas tecnologias. Encerra-se o Capítulo, descrevendo do paradigma SDN e respectivas soluções para aplicá-lo ao cenário de Redes Sem Fio. No Capítulo 3 serão revistas as principais topologias de Redes Sem Fio para, logo em seguida, apresentar o FRESDWN. Será apresentada sua ideia principal e a arquitetura do *FRESDWN*. No Capítulo 4 serão detalhados os testes de

validação da proposta e serão apresentadas as limitações da implementação. O Capítulo 5 finaliza este trabalho, trazendo a conclusão da dissertação e os trabalhos futuros.

2 Revisão da Literatura

Este Capítulo tem como objetivo apresentar ao leitor as informações necessária ou, mais especificamente, a fundamentação teórica a respeito da subárea de conhecimento que esta dissertação se propõe a investigar e apresentar uma solução. Desta forma, a Seção 2.1 descreve, de forma geral, as Redes Definidas por Software e na Seção 2.2 são apresentados os principais trabalhos relacionados a Redes Sem Fio Definidas por Software.

2.1 Redes Definidas por Software

O conceito de Redes Programáveis surgiu no final do século passado, propondo uma clara separação do plano de controle do plano de dados, por uma API bem definida, ao contrário da integração vertical entre os planos que acontecem tradicionalmente nos roteadores e comutadores (CAMPBELL et al., 1999).

As Redes Definidas por Software são uma continuação deste conceito de Redes Programáveis e, portanto, vem a romper algumas limitações das redes tracionais ao permitir contornar restrições em relação à inovação, ao controle do desempenho e também de configuração, cujas principais diferenças são apresentadas na Tabela 1.

Tabela 1 – Comparativo entre SDN e redes tradicionais, baseado em (XIA et al., 2015a)

	Redes Definidas por Software	Redes Tradicionais
Características	Desacoplamento dos plano de controle e de dados, sendo programável	Um novo protocolo construído para atender a um problema específico, o controle da rede é complexo
Configuração	Configuração automatizada e validação centralizada	Detectação e mitigação de erros de configuração é manual
Desempenho	Controle dinâmico e global, utilizando informações de várias camadas de redes	Informação limitada e configurações relativamente estáticas
Inovação	Facilidade de implementação de novos conceitos em *software*, ambiente suficientemente isolado para efetuar testes e permitir a implantação incremental ou rápida, através de uma simples atualização de *software*	Dificuldade de implementar novos conceitos em *hardware*, ambiente de testes bem restrito e a implantação demanda um longo processo de padronização

Campbell (CAMPBELL et al., 1999) define que a arquitetura de redes tem que possuir os seguintes atributos:

– Serviços de rede, que a arquitetura considera um conjunto de algoritmos distribuídos e oferecidos aos dispositivos de usuários;

– Algoritmos de rede, que incluem o transporte, e os mecanismos de controle, sinalização e gerência;

– Várias escalas de tempo, o que impacta na influência do projeto de rede;

– Gerência do estado da rede, que inclue o estado que os algoritmos de rede operam para garantir a consistência dos serviços (roteamento, comutação, dentre outros).

A programabilidade de redes oferecida pelo paradigma de Redes Definidas por Software (SDN) traz várias inovações. Transforma a idéia tradicional em redes abertas, flexíveis e de rápida evolução. Isto porque, através de uma interface padronizada e um sistema de controle externo, fornece um gerenciamento mais eficiente dos elementos da rede(HALEPLIDIS et al., 2015a).

A RFC 7426 (HALEPLIDIS et al., 2015a), através do grupo SDNRG (*IRTF Software-Defined Networking Research Group*) da IRTF (*Internet Research Task Force*), padronizou as terminologias das camadas e da arquitetura do SDN, com o intúito de servir de base para discussões futuras, conforme pode ser ilustrado pela Figura 1.

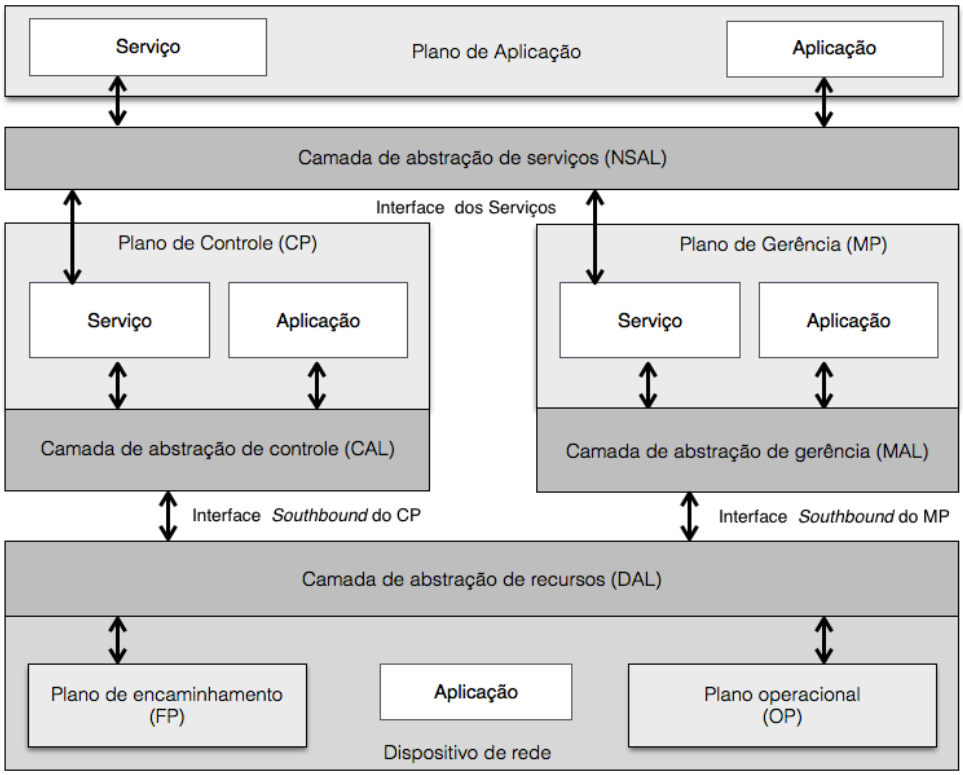

Figura 1 – A arquitetura SDN adaptada de (HALEPLIDIS et al., 2015a)

A seguir, são detalhadas essas terminologias e suas características principais:

- Redes Definidas por Software ou SDN (*Software-Defined Networking*) - uma abordagem de rede programável que suporta a separação do plano de controle do plano de encaminhamento de dados, através de uma interface padronizada.

- Recurso - Um componente físico ou virtual disponível no sistema. Um recurso pode ser bem simples (porta ou fila) ou complexo, quando é composto por múltiplos recursos (um dispositivo de rede).

- Dispositivo de rede - Um dispositivo que executa uma ou mais operações de redes relacionadas à manipulação e encaminhamento de pacotes. Este modelo de referência não faz nenhuma distinção se o dispositivo é físico ou virtual e tanto pode ser implementado em *hardware* quanto *software*. Um dispositivo pode ser considerado um *container* de recursos, e, como descrito acima, pode ser um recurso complexo. Exemplo de dispositivos de rede são *switches* e roteadores, mas esses dispositivos podem operar em camadas acima do IP (*firewall*, balanceadores de carga, codificadores de vídeo, dentre outros) ou abaixo do IP (como *switches* L2 (*Layer 2*), elementos de rede ótica e de microondas). Cada dispositivo de rede tem uma presença nos planos FP e OP.

- Interface - O ponto de interação entre duas entidades. Conforme será detalhado abaixo e, ilustrado na Figura 1, todos os planos são conectados por interfaces. Uma interface pode possuir múltiplos papéis, dependendo se os planos conectados residem no mesmo dispositivo, independente de ser físico ou virtual. Quando as entidades são dispostas em localizações distintas, a interface é usualmente implementada através do protocolo de rede. Se as entidades estão no mesmo local físico, a interface pode ser implementada com código proprietário ou livre, utilizandoo uma API (*Application Programming Interface*), ou IPC (*Inter-Process Communication*), ou um protocolo de rede.

  - interface CPSI (*Control-Plane Southbound*[1] *Interface*) é uma interface de tempo crítico que requer uma baixa latência, e algumas vezes, grande largura de banda, para permitir executar várias operações rapidamente. Um exemplo, são atualizações rápidas e frequentes nas tabelas ou fluxos, alta vazão, robustez da manipulação de pacotes e eventos. Implementações de protocolos desta interface são ForCES (DORIA et al., 2010) e Openflow (MCKEOWN et al., 2008).
  - interface MPSI (*Management-Plane Southbound Interface*) não é, usualmente, uma interface sensível ao tempo e, sim, mais voltada para a interação humana, orientada à usabilidade e suas mensagens são menos frequentes que na CPSI.

[1] Southbound é a interface que permite a efetiva comunicação entre a controladoras e os nós pelo protocolo Openflow. Northbound é a interface que permite a comunicação de aplicações das camadas superiores com a controladora.

Segundo a RFC 3535 (SCHOENWAELDER, 2003) que preconiza que as tecnologias de gerência de rede devem ser de fácil utilização e não necessariamente performáticas. O MPSI pode ser um protocolo, uma API ou um IPC. Se o MP não for embutido no dispositivo de rede, o MPSI certamente é um protocolo.

– Aplicação ou App (*Application*) - Uma aplicação no contexto do SDN é um pedaço do software que utiliza os serviços das camadas abaixo para executar determinada função, sem prover acesso para outras aplicações e pode ser implementado em um plano ou em múltiplos planos. A operação de uma aplicação pode ser parametrizada, por exemplo, ao passar determinados argumentos em tempo de chamada, mas é planejada para ser um software *standalone*. Uma aplicação não oferece nenhuma interface para outras aplicações ou serviços. Caso as aplicações se espalhem por múltiplos planos, podem utilizar a interface *southbound* do CP ou CPSI (*Control-Plane Southbound Interface*) e, também, utilizar a interface *southbound* do MP ou MPSI (*Management-Plane Southbound Interface*), o que está ilustrado na Figura 1.

– Serviço - Um pedaço do software que executa uma ou mais funções e provê uma ou mais APIs para as aplicações ou outro serviço da mesma ou de camadas distintas para se utilizar das suas funções e retornar um ou mais resultados. Serviços podem ser combinados com outros serviços, ou chamados de uma maneira serial para criar um novo serviço. Podem ser espalhados por múltiplos planos e pertencem também ao plano de aplicação, conforme ilustrado na Figura 1.

– Plano de encaminhamento ou FP (*Forwarding Plane*) - É o conjunto de recursos espalhado por todos os dispositivos de rede responsáveis pelo encaminhamento do tráfego de dados, também conhecido como plano de dados (*data plane*), ou caminho de dados (*data path*). É responsável por gerenciar os pacotes no fluxo de dados, baseado nas instruções recebidas do CP. Ações no plano de encaminhamento incluem o encaminhamento, descarte e alterações de pacotes e podem conter recursos de encaminhamento como classificadores. O plano de encaminhamento é, normalmente, o ponto terminal do serviço e aplicações do CP.

– Plano operacional ou OP(*Operational Plane*) - É o conjunto de recursos responsável pela gerência do estado operacional individual de todos os dispositivos de rede, ou seja, quando o disposivo está ativo ou inativo, número de portas disponíveis, o *status* de cada porta, memória, dentre outros. O OP é ponto terminal das aplicações e serviços do MP, pois relaciona os recursos dos dispositivos de rede. No SDNRG da IRTF alguns acreditam que o OP não constitui um plano e, sim, é a união das funções do FP, mas a maioria aceita que, por permitir distinguir entre as diferentes áreas de operação, deve ser classificado como um plano, conforme a Figura 1.

– Plano de controle ou CP (*Control Plane*) - É o conjunto de funções responsável por controlar um ou mais dispositivos de rede. É responsável por tomar as decisões de como os pacotes devem ser encaminhados para um ou vários dispositivos de rede e transmitir essa decisão para que os dispositivos de rede a executem. O CP interage primariamente com o FP e, em menor extensão, com o OP, e pode estar interessado pelas informações do OP que podem ser o estado atual de uma porta particular ou de suas capacidades. As funcionalidades do CP incluem:

  – Descoberta e manutenção da topologia de rede.
  – Instanciação e seleção de rotas de pacotes.
  – Mecanismos de tolerância a falha de rotas.

  Em suma, o CP informa, ao dispositivo de rede, o modo que o mesmo deve processar e encaminhar os pacotes, efetuando o ajuste fino na tabela de encaminhamento do FP, baseado na topologia da rede ou de requisições de serviços externos.

– Plano de gerência ou MP (*Management Plane*) - É o conjunto de funções responsável por monitorar, configurar e manter um ou mais dispositivos de redes ou partes de dispositivos de rede, como, por exemplo, tomar decisões baseadas nos estados dos dispositivos de rede. As funcionalidades do MP são tipicamente iniciadas e baseadas na visão geral da rede, tradicionalmente centrada no ser humano. Apesar disso, ultimamente, algoritmos têm substituído a maior parte da intervenção humana, também chamados de orquestradores (*orchestrators*) e gestores virtuais (*VNF Managers* (*Virtual Network Function Managers*) e *Virtualised Infrastructure Managers*). As funcionalidades típicas são:

  – Gerência e monitoramento de falha;
  – Gerência de configuração.

  O MP é, predominantemente, relacionado com o OP e menos relacionado com o FP, pois, em casos raros, o MP pode configurar todas ou parte das regras de encaminhamento de uma só vez.

– Plano de aplicação - É o conjunto de aplicações e serviços que programam o comportamento da rede, ou seja, definem o comportamento da rede. As aplicações e os serviços que utilizam serviços do CP e MP compõem o plano de aplicações. Aplicações que diretamente ou primariamente suportam operações do plano de encaminhamento (como o processo de roteamento no plano de controle) não são consideradas parte do plano de aplicação. Exemplos de aplicações incluem descoberta de topologia de rede, provisionamento da rede, reserva de caminho de rede, dentre outras. Vale destacar que a aplicação pode ser implementada de maneira modular e distribuída e, portanto, pode se espalhar por múltiplos planos, conforme a Figura 1.

- Camada de abstração de recursos ou DAL (*Device and resource Abstraction Layer*) - A DAL é baseada em um ou mais modelos. Abstrai os recursos dos dispositivos dos planos FP e OP para os planos CP e MP. Se é um dispositivo físico, o mesmo pode ser referenciado como HAL (*Hardware Abstraction Layer*). Em suma, o DAL provê um ponto de referência uniforme para os recursos de dispositivos dos FP e OP.

- Camada de abstração de controle ou CAL (*Control Abstraction Layer*) - A CAL provê acesso para aplicações e serviços do CP, abstraindo a interface *Southbound* do CP. Aplicações de controle podem utilizar a CAL para controlar dispositivos de rede, sem prover serviços para as camadas superiores (OSPF, IS-IS e BGP são exemplos dessas aplicações). Como exemplos de serviços do CP, podemos destacar PVLAN (*Virtual Private LAN*), tunéis e serviços de topologia.

- Camada de abstração de gerência ou MAL (*Management Abstraction Layer*) - A MAL provê acesso para aplicações e serviços do MP, abstraindo a interface *Southbound* do MP (MPSI). As aplicações de gerência podem utilizar a MAL para gerir os dispositivos de rede, sem prover qualquer serviço às camadas superiores. Os serviços do MP provêem acesso a outros serviços e aplicações de camandas acima do MP, conforme a Figura 1. Exemplos de aplicações de gerência estão o monitoramento de rede, detecção de falhas e aplicações de recuperação.

- Camada de abstração de serviços ou NSAL (*Network Services Abstraction Layer*) - Provê abstrações que podem ser utilizadas tanto por aplicações ou por outros serviços para acessar serviços dos planos de aplicação, controle e gerência. Exemplos dessas interfaces de serviços são: RESTful APIs, protocolos abertos como NETCONF, IPC, CORBA, dentre outras. As duas interfaces mais utilizadas são: RESTful e RPC(*Remote Procedure Call*). Ambas obedecem uma arquitetura cliente/servidor e trocam mensagens através de XML (*eXtensible Markup Language*) ou JSON (*JavaScript Object Notation*). A seguir, seguem as principais características de cada uma dessas interfaces.

  A interface RESTful, baseada no paradigma REST (*representational state transfer*), possui as seguintes caracterísiticas:

  - Identificação dos recursos - recursos individuais são identificados utilizando um identificador de recursos, como por exemplo o URI (*Uniform Resource Identifiers*);
  - Manipulação dos recursos através das representações - recursos são representados em um formato como JSON, XML, ou HTML(*HyperText Markup Language*);
  - Mensagens auto-descritivas - Cada mensagem tem informação suficiente para descrever como a mensagem deve ser processada;

  - Hipermídia como o motor do estado da aplicação - O cliente não precisa de nenhum conhecimento anterior de como interagir com o servidor, pois a API não é fixa e sim dinamicamente gerada pelo servidor.

A interface das RPCs, como por exemplo a XML-RPC, possui as seguintes características:

  - Procedimentos individuais são identificados através de um identificador;
  - O cliente precisa conhecer o nome do procedimento e os parâmetros associados ao mesmo.

Nas redes tradicionais, o CP é usualmente implementado acoplado ao dispositivo de rede, tornando-o distribuído por toda a rede, enquanto o MP é tradicionalmente centralizado e é responsável por gerenciar os dispositivos e o CP. Entretanto, com a adoção do SDN essa distinção não se mantêm tão clara.

Em relação ao quesito tempo, de modo geral, o CP precisa enviar atualizações frequentes, o que é algo na faixa dos milissegundos, o que precisa de alta largura de banda e baixa latência dos *links*. Em contra partida, o MP necessita de tempos de reação geralmente mais longos, podendo ser na ordem de minutos, horas ou até dias. Outra distinção entre o CP e o MP é no quesito persistência do estado. O CP utiliza persistências efêmeras, ou seja, possuem um ciclo de vida bem curto e podem ser armazenadas em memórias voláteis (por exemplo, a determinação de rotas). Por outro lado, o MP usualmente utiliza estados persistentes com um longo ciclo de vida, que pode variar de horas, dias ou meses o que significa que é utilizado além do ciclo de vida do processo que o criou.

Vale destacar que, apesar de não estar ilustrado na Figura 1, serviços, aplicações e até mesmo planos completos podem ser dispostos de uma maneira recursiva, provendo uma semântica de *overlay* para o modelo. Por exemplo, um serviço do plano de aplicação pode prover para outro aplicativo ou serviço através do NSAL.

Apesar de existirem várias implementações de SDN, destacaremos a seguir, algumas das mais relevantes e motivaremos a escolha do Openflow para o FRESDWN.

2.1.1 ForCES

O IETF (*Internet Engineering Task Force*), em um esforço de padronizar o ForCES (*Forwarding and Control Element Separation*), publicou a especificação detalhada da arquitetura do *framework* modular do sistema que implementa a separação dos dois planos de maneira escalável, flexível, totalmente programável e independente do vendedor. No modelo do ForCES cada elemento de rede é composto de várias entidades com funcionalidades bem definidas, que cooperam para prover a funcionalidade desejada, como roteamento e encaminhamento IP. O modelo se baseia no fato de que a implementação do

elemento de rede, habilitado pelo ForCES, seria indistinto do elemento de rede tradicional, o que facilitaria a migração para a nova arquitetura, permitindo que seja implementado na infraestrutura atual e em infraestruturas experimentais ou em implantação. A falta de uma rede programável, com uma abordagem clara e uniforme, ainda é um desafio que deve ser superado (HALEPLIDIS et al., 2015b).

O *framework* ForCES é baseado no dispositivo de rede, conforme pode ser observado na Figura 2, tendo como objetivo separar o CP do FP, o que é obtido pela separação dos protocolos. De modo mais conceitual, opera no modo mestre/escravo, sendo os elementos de controle os mestres dos elementos de encaminhamento. A funcionalidade do plano de controle ou "*slow path*", que é exemplificada pelos protocolos de roteamento, protocolos de sinalização e pelo controle de admissão, é separada da funcionalidade do plano de encaminhamento ou "*fast path*", que é uma atividade por pacote de dados e exemplificada pelo encaminhamento de pacotes propriamente ditos, modificação de cabeçalho e das suas filas. No *framework* o dispositivo de rede é composto em CEs (*Control Elements*) e FEs (*Forwarding Elements*). Inclui comandos para envio de informações de configuração, associação, estado do dispositivo, bem como suporte a eventos e notificações. Para o CE controlar qualquer FE foi decidido uma abstração no modelos para as FEs, focando no FP, o que permite ao desenvolvedor definir os seus modelos de abstração para o FE, através da linguagem de modelagem descrita no modelo ForCES (YANG et al., 2004).

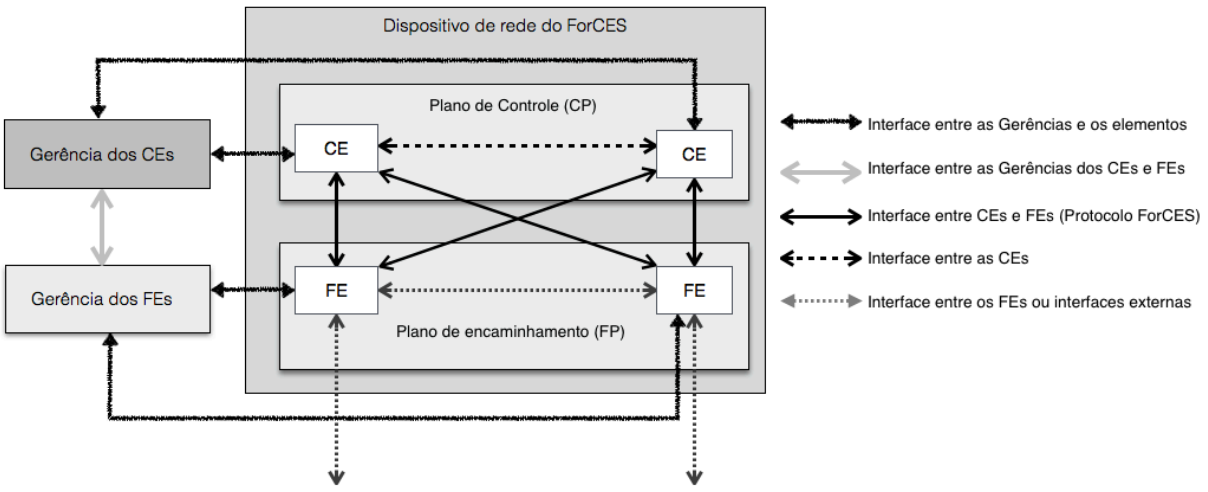

Figura 2 – A arquitetura ForCES adaptado de HALEPLIDIS et al., 2015b

Comparando a Figura 2 com a Figura 1, podemos verificar que o modelo ForCES provê uma linguagem de modelagem que abstrai os recursos do dispositivo de rede e, portanto, atua como o DAL e, através da interface do protocolo ForCES, é possível que os recursos abstraídos sejam controlados e gerenciados pelo CPSI, apesar de também poder ser aplicada a MPSI. Como ilustrado na Figura 2, o *framework* inclui duas entidades que são: Gerência dos CEs ou CEM (*CE Manager*) e a Gerência dos FEs ou FEM (*FE*

Manager). O CEM e o FEM são responsáveis por determinar quais FEs poderam ser associadas com quais CEs e instruí-las com os detalhes necessários a associação como os elementos IPs, previamente a qualquer comunicação entre os CE e o FE.

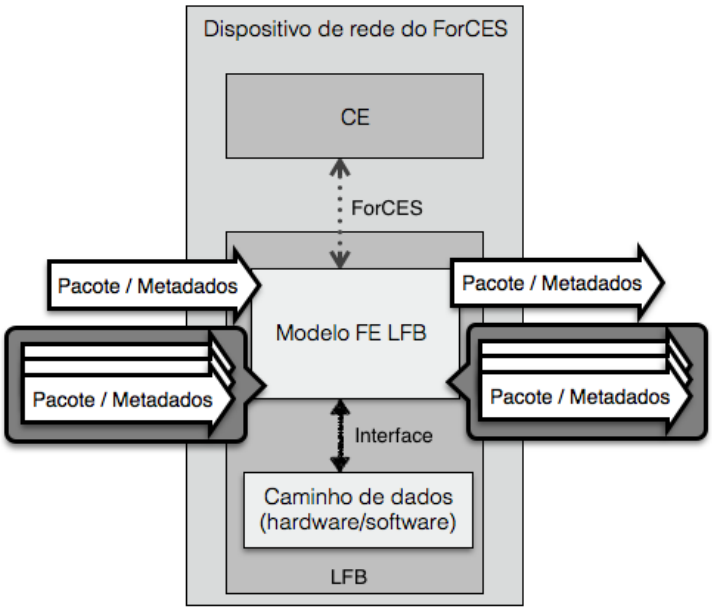

Figura 3 – A arquitetura de abstração do FE e entrada e saída do LFB do ForCES adaptada de HALEPLIDIS et al., 2015b

A Figura 3 ilustra a visão de alto nível das operações do ForCES, compreendendo a interface entre o modelo do FE e a implementação do caminho de dados em *hardware* ou *software*, que pode ser feita em uma API aberta, ou uma interface de *Kernel* ou até mesmo uma interface proprietária. Para o CE efetivamente controlar a FE, é necessário entender como o FE processa os pacotes. O LFB (*Logical Functional Blocks*) são abstrações de diferentes elementos que efetuam ações ou manipulações específicas nos pacotes que passam por ele. Após completar a função associada ao LFB, ou o pacote foi modificado de determinada maneira, ou algum resultado foi gerado ou transmitido, normalmente sobre a forma de matadados. Classificadores, modeladores e medidores são exemplos de LFB, cujas entradas e saídas de dados/metadados do LFB são ilustradas na Figura 3. Metadados são dados que foram criados pelo LFB para facilitar o próximo LFB, na sua sequência do grafo de encaminhamento para processar o pacote.

Na RFC 3746 YANG et al., 2004 é apresentado um modelo e dois protocolos. o FP é separado do CP por uma interface conhecida como protocolo ForCES DORIA et al., 2010, que opera na entidade do FP e foi modelada utilizando o modelo ForCES HALPERN; SALIM, 2010. O modelo ForCES é baseado no fato que o elemento de rede é composto de várias entidades logicamente separadas, que cooperam para prover uma determinada funcionalidade (como roteamento ou comutação IP) e, mesmo assim, aparecem como num elemento de integração de rede para a entidade externa. O modelo ForCES o FP utiliza os LFBs (*Logical Functional Blocks*), que quando conectados a um grafo, compõe o FE

(*Forwarding Element*). Os LFBs são descritos em XML. As definições do LFB incluem: tipos de dados básicos e customizados, definição de metadados, portas de entrada e saída, parâmetros e componentes operacionais e definições de eventos. O modelo permite definir LFBs físicas ou virtuais de pequena ou grande granularidade. O protocolo ForCES é agnóstico e pode ser utilizado para monitorar, configurar e controlar qualquer elemento do modelo. O protocolo possui comandos simples para: atribuir, obter e apagar. O protocolo ForCES é projetado para obter atualizações rápidas e grande largura de banda. Com base na Figura 1, o modelo ForCES é aplicável ao DAL, tanto para o OP quanto o FP, utilizando LFBs. O protocolo ForCES é projetado e aplicável ao CPSI, e também, pode ser utilizado pelo MPSIHALEPLIDIS et al., 2015a.

Até o presente momento, apesar de todos os esforços do IETF, o ForCES só despertou algum interesse da comunidade científica, o que infelizmente não aconteceu fora da academia, em especial na indústria. O ForCES e o Openflow consideram o dispositivo de rede simples ou "burro", com a separação do plano de controle. Uma principal diferença entre eles é a flexibilidade do ForCES. Enquanto o Openflow está limitado ao controle do caminho de dados a lista de características que o dispositivo provê de acordo com a versão do protocolo suportada. Dois pontos negativos do ForCES - ficou fora do escopo as redes sem fio e não se preocupou com aspectos de endereçamento como Unicast, Multicast e Broadcast.(HALEPLIDIS et al., 2012,HALEPLIDIS et al., 2015b).

2.1.2 SANE e Ethane

O SANE CASADO et al., 2006 utilizava redes baseadas em fluxos e faziam uso do controlador de domínio que tratava de autenticações, políticas de segurança globais e também decidia o processamento dos *switches* em relação aos fluxos. O controlador é responsável pela gerência de políticas e, também, pela segurança em uma rede. Define que todas as decisões de roteamento e controle de acesso são feitas por um servidor logicamente centralizado, que concede acesso aos serviços com a distribuição de recursos de acordo com as políticas de controle de acesso. A mobilidade dos clientes dentro da LAN é transparente para os servidores, porque o serviço é desconhecido. Quando um cliente altera a sua posição,como por exemplo, move-se para um diferente ponto de acesso sem fio (AP), o mesmo atualiza as suas capacidades e passa novas rotas de retorno para os servidores que estiverem acessando. Se um cliente se move, o mesmo deve revogar o seu conjunto atual, evitando que uma nova máquina ligada ao mesmo ponto de acesso possa acessar o tráfego enviado para o cliente anterior, depois que o mesmo deixou o AP. Em suma, buscava melhorar a segurança da rede através da centralização das decisões de encaminhamento e acesso ao utilizar um servidor logicamente centralizado.

Ethane CASADO et al., 2007 foi uma implementação concreta com base na filosofia do SANE, também criada pelos mesmos pesquisadores. O Ethane foi aplicado em

dois componentes: um controlador que gerencia a admissão e o chaveamento de fluxos, decidindo se um pacote deve ser encaminhado, e um *switches* simples, que é composto de uma tabela de fluxos e um canal seguro para comunicação com o controlador, sendo este o responsável pelo encaminhamento dos pacotes. A arquitetura do Ethane é baseada em três princípios:

- a rede deve ser gerenciada através de políticas de alto nível;
- a política deve indicar qual o melhor caminho para os pacotes;
- a rede deve ser capaz de rastrear a origem dos pacotes.

Para atender a esses princípios, o Ethane emprega um controlador que utiliza uma política global da rede, na qual quaisquer comunicações entre equipamentos devem ser explicitamente permitidas pelo controlador. Dessa forma, o controlador pode calcular as rotas para os fluxos permitidos, utilizando informações da topologia da rede. Cada elemento de rede deve consultar a controladora, ao identificar um novo fluxo; esta, por sua vez, consulta as políticas globais para decidir, com base nas características de cada fluxo, como o elemento de encaminhamento deveria tratá-lo. Essa decisão é comunicada ao *switch*, que constroi uma regra adequada para o novo fluxo, através de uma entrada em sua tabela de encaminhamento. Apesar da nova abordagem, este projeto era compatível com as máquinas (*nós*) e *switches* existentes (*Backward Compatibility*).

2.1.3 Openflow

O protocolo Openflow[2] é uma das implementações do conceito de SDN, apresentado em (MCKEOWN et al., 2008) e tem origem nos trabalhos SANE (CASADO et al., 2006) e Ethane (CASADO et al., 2007). A possibilidade de gerir redes com base nesta filosofia exigia um protocolo específico para a comunicação entre os *switches* e os controladores. Openflow (MCKEOWN et al., 2008) é uma especificação pública, inspirada no Ethane, e permite múltiplas formas de gerir a rede e novas aplicações.

A abordagem SDN acrescenta flexibilidade ao contexto do encaminhamento de dados, mas, na maioria das vezes, os programas das controladoras são executados em servidores de uso geral, através de linguagens de alto nível, como C++ ou PYTHON.

Quando foi criado o Openflow, os quatro objetivos idealizados eram:

- suportar equipamentos de alto desempenho;
- baixar os custos pela normalização do Openflow;

[2] Openflow é um padrão aberto que permite aos pesquisadores executar protocolos experimentais nas redes das suas instituições, podendo ser utilizado em *switch*es e roteadores Ethernet comerciais e também pontos de acesso sem fio. <https://www.opennetworking.org/>

– suportar uma larga escala de investigações nesta área;

– isolar o tráfego experimental do tráfego de produção.

2.1.3.1 Características Básicas

O Openflow fornece uma interface southbound para a arquitetura SDN. O Openflow é um protocolo orientado a fluxo e possui abstrações para os *switch*es e suas portas, usadas para controlar o fluxo de dados.

As Redes Definidas por Software, particularmente o Openflow, possui alguns desafios. A separação do plano de dados do plano de controle, em geral, tende a piorar desempenho, em termos de atrasos adicionais para operações de controle, como no caso das configurações de fluxos pelo controlador para a descoberta de topologia e na recuperação de falhas. Além disto, a centralização do plano de controle insinua a possibilidade de problemas com a escalabilidade e com a confiabilidade do controlador, criando o dilema da existência de um ponto de falha único, com as dificuldades adicionais na implementação de um sistema de controle distribuído. Em uma rede atual de alto desempenho e com muitos usuários, temos que considerar três elementos: o desempenho, a capacidade de programação e a flexibilidade. O desempenho refere-se, especificamente, à velocidade de processamento do nó de rede, considerando tanto a vazão quanto a latência. A utilização de uma abordagem SDN implica no acréscimo de uma latência adicional, resultante da tomada de decisão pelo controlador, caso a regra não esteja programada no dispositivo.

2.1.3.2 Controladora Openflow

As controladoras são responsáveis por programar as tabelas de fluxos dos *switch*es. Existem diversos tipos de controladoras SDN, como o Beacon[3], Floodlight[4], Maestro[5], NOX[6], OpenDayLight[7], POX[8], RYU[9], Trema[10], dentre outras.

[3] Beacon é uma controladora openflow desenvolvida em JAVA pela universidade de Stanford que é multi-plataforma, modular, rápida e suporta operações baseadas em theads e a eventos. <https://openflow.stanford.edu/display/Beacon/Home>

[4] Floodlight é uma controladora openflow desenvolvida em JAVA que é patrocinada pela Big *switch* Networks. <http://www.projectfloodlight.org/floodlight/>

[5] Maestro é uma controladara Openflow baseada em Java que também pode funcionar como um orquestrador, ou seja, uma plataforma para alcançar funções de controle automático da rede e programá-las através de seus módulos. <https://zhengcai.github.io/maestro-platform/>

[6] NOX é uma controladora Openflow para gerenciamento de redes de grande escala desenvolvido em PYTHON/C++ pela NICIRA, sendo um dos primeiros controladores OpenFlow. <https://github.com/noxrepo/nox>

[7] OpenDayLight é uma controladora que suporta o OPNFV, que é um projeto de código aberto voltada para a evolução do NFV através de uma plataforma integrada e aberta. <https://www.opendaylight.org/lithium>

[8] POX é uma controladora Openflow para gerenciamento de redes de grande escala desenvolvido em PYTHON pela NICIRA. <http://www.noxrepo.org./pox/about-pox/>

[9] RYU é uma controladora baseada em PYTHON extremamente flexível que possui um amplo suporte ao OpenFlow (1.0, 1.2, 1.3, 1.4, 1.5 e as extensões Nicira). <https://osrg.github.io/ryu/>

[10] Trema é uma controladora baseada em Ruby. <http://trema.gitub.io/trema/>

Nesta dissertação iremos utilizar a controladora RYU, por ser multiprotocolo e, também, ser a controladora com melhor suporte a novas versões do protocolo Openflow.

2.1.3.3 *switch* Openflow

Os *switches* compatíveis com Openflow podem ser separados em dois tipos principais: Openflow-exclusivo e Openflow-híbrido (Foundation [2013]). Os *switches* Openflow-exclusivo fornecem apenas operações definidas no protocolo Openflow, ou seja, todo o fluxo de pacotes é processado pelo controlador Openflow. *switches* híbridos suportam o protocolo Openflow (em algum nível), mas também realizam operações normais de comutação em Camada 2 (Enlace) e/ou Camada 3 (Rede) (MCKEOWN et al., 2008).

Dentre as diversas implementações de *switches* Openflow, podemos destacar uma implementação de *switch* Openflow a nível de núcleo do sistema chamada Open v*switch* (OVS)[11], e uma implementação a nível de biblioteca de usuário chamada OpenFlow 1.3 Software *switch* (ofsoft*switch*13)[12].

2.1.3.4 Mensagens Openflow

Cada mensagem Openflow é trocada entre a controladora e o cliente, através de um canal seguro e confiável e possui três campos básicos que são: regra, ação e estatísticas, como pode ser visualizado na Figura 4. As regras são características de redes provenientes de diversas camadas da pilha de protocolos e são responsáveis por correlacionar as informações da unidade de dados com o o comportamento programado pela controladora para o cliente Openflow, de acordo com as informações de fluxo de dados, recebidos. As ações são responsáveis por definir o que deve ser feito com o fluxo de dados que coincide com o atrelado no campo regras. As estatísticas permitem analisar e avaliar métricas dos fluxos.

A seguir, podemos observar na Tabela 2 e na Tabela 3, que ocorre uma significativa evolução entre algumas das versões do protocolo Openflow, em especial aos tipos de mensagens suportadas por cada versão. A Tabela 2 apresenta todas as mensagens prevista na versão inicial do protocolo Openflow e seus códigos equivalentes nas demais versões. A Tabela 3 continua a partir do código não previsto na versão inicial do Openflow e detalha todos os demais. Em relação à compatibilidade entre versões do protocolo Openflow, notamos que algumas mensagens foram suprimidas e outras trocaram de código identificador entre versões. Essas alterações implicam em efetuar um conjuntos de ajustes para compatibilizar o que foi desenvolvido para que uma determinada versão possa ser utilizada

[11] Open v*switch* é um *switch* openflow de qualidade de ambiente de produção, possui *switch*es virtuais de múltiplas camadas.

[12] Open v*switch* é um *switch* openflow implementação no endereçamento de usuário e suporta o OpenFlow 1.3. O código é baseado no *switch* Ericsson TrafficLab 1.1 Soft*switch*, com alterações no plano de encaminhamento para suportar OpenFlow 1.3.

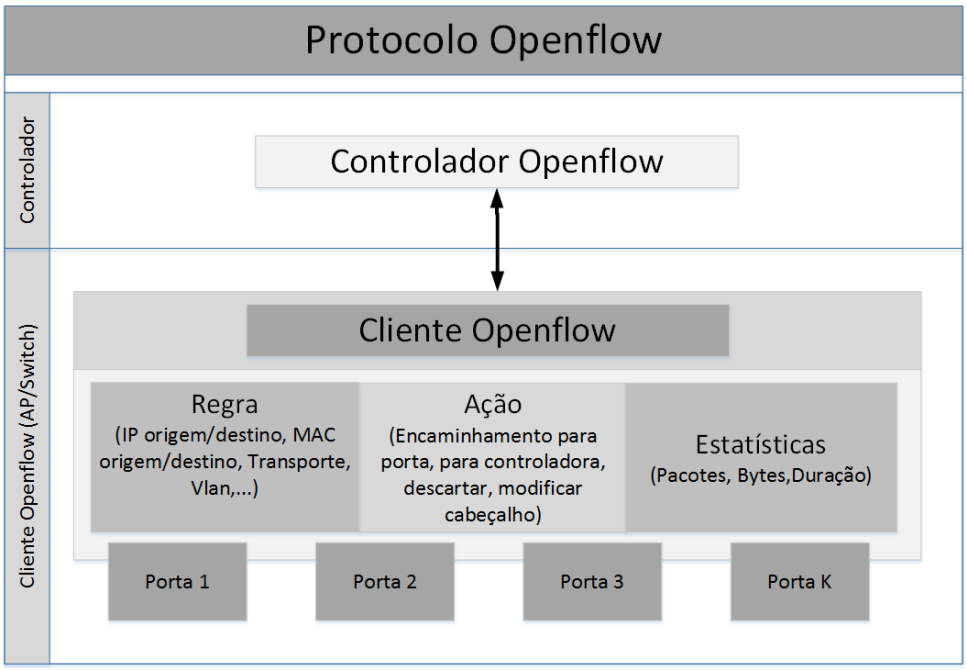

Figura 4 – Ilustra a comunicação entre os nós sem fio móvel e a controladora

em uma mais recente. Vale destacar que, com o *framework* proposto por este trabalho, permitiremos a troca de mensagens simétricas do tipo Experimenter, ou seja, do tipo 4. A mensagem Experimenter[13] se manteve constante com o tipo 4 em todas as versões, sendo que, inicialmente, esse tipo de mensagem era conhecida como Vendor[14]. Após as versões iniciais do Openflow, essa mensagem manteve a sua estrutura de dados constante em todas as versões. Essa mensagem é indicada como a forma padrão para os *switches* OpenFlow oferecerem funcionalidades adicionais daquelas suportadas nativamente pelo protocolo, o que torna essa mensagem uma candidata ideal para inovar.

[13] Experimenter é uma mensagem que pode ser utilizada pelo pesquisador ou "fornecedor". A partir da da versão 1.2 não sofreu alteração e possui os seguintes campos: um cabeçalho, um campo de 4 bytes que é o identificador do experimentador,um campo de 4 bytes do tipo de pesquisador e de dados definidos pelo usuário com um comprimento arbitrário.

[14] Vendor é uma mensagem de "fornecedor"da versão 1.0, e possui os seguintes campos: um cabeçalho, a identificação do fornecedor de 4 bytes e dados definidos pelo usuário com um comprimento arbitrário. Quando o byte mais significativo for 0, os próximos três bytes são o identificador do fornecedor (IEEE OUI).

Tabela 2 – Comparativo entre os tipos das mensagens em cabeçalho Openflow (até o suportado pelo Openflow 1.0)

Versão	1.0	1.1	1.2	1.3	1.4	1.5	Mensagem
Cód. Tipo	1	2	3	4	5	6	-
0	hello	hello	hello	hello	hello	hello	simétrica[15]
1	error	error	error	error	error	error	simétrica
2	echo request	echo request	echo equest	echo request	echo request	echo request	simétrica
3	echo reply	echo reply	echo reply	echo reply	echo reply	echo reply	simétrica
4	vendor	experimenter	experimenter	experimenter	experimenter	experimenter	simétrica
5	features request	features request	features request	features request	features request	features request	da controladora para o *switch*
6	features reply	features reply	features reply	features reply	features reply	features reply	da controladora para o *switch*
7	get config request	get config request	get config request	get config request	get config request	get config request	da controladora para o *switch*
8	get config reply	get config reply	get config reply	get config reply	get config reply	get config reply	da controladora para o *switch*
9	set config	set config	set config	set config	set config	set config	da controladora para o *switch*
10	packet in	packet in	packet in	packet in	packet in	packet in	assíncrona
11	flow removed	flow removed	flow removed	flow removed	flow removed	flow removed	assíncrona
12	port status	port status	port status	port status	port status	port status	assíncrona
13	packet out	packet out	packet out	packet out	packet out	packet out	da controladora para o *switch*
14	flow mod	flow mod	flow mod	flow mod	flow mod	flow mod	da controladora para o *switch*
15	port mod	group mod	group mod	group mod	group mod	group mod	da controladora para o *switch*
16	stats request	port mod	port mod	port mod	port mod	port mod	da controladora para o *switch*
17	stats reply	table mod	table mod	table mod	table mod	table mod	da controladora para o *switch*
18	barrier request	statsreq	stats request	multipart request	multipart request	multipart request	da controladora para o *switch*
19	barrier reply	stats reply	stats reply	multipart reply	multipart reply	multipart reply	da controladora para o *switch*
20	queue get config request	barrier request	barrier request	barrier request	barrier request	barrier request	da controladora para o *switch*
21	queue get config reply	barrier reply	barrier reply	barrier reply	barrier reply	barrier reply	da controladora para o *switch*

[15] simétrica - caracterizamos as mensagens openflow simétricas como aquelas que são enviadas sem solicitação, em ambas direções. Podem ser geradas tanto pelo controlador quanto pelo *switch*. São enviadas sem solicitação.

Tabela 3 – Comparativo entre os tipos das mensagens em cabeçalho Openflow (após o suportado pelo Openflow 1.0)

Versão	1.0	1.1	1.2	1.3	1.4	1.5	Mensagem
Cód. Tipo	1	2	3	4	5	6	-
22	-	queue get config request	queue get config request	queue get config request	-	-	da controladora para o *switch*
23	-	queue get config reply	queue get config reply	queue get config reply	-	-	da controladora para o *switch*
24	-	-	role request	role request	role request	role request	da controladora para o *switch*
25	-	-	role reply	role reply	role reply	role reply	da controladora para o *switch*
26	-	-	-	get async request	get async request	get async request	da controladora para o *switch*
27	-	-	-	get async reply	get async reply	get async reply	da controladora para o *switch*
28	-	-	-	set async	set async	set async	da controladora para o *switch*
29	-	-	-	meter mod	meter mod	meter mod	da controladora para o *switch*
30	-	-	-	-	role status	role status	assíncrona[16]
31	-	-	-	-	table status	table status	assíncrona
32	-	-	-	-	request forward	request forward	assíncrona
33	-	-	-	-	bundle control	bundle control	da controladora para o *switch*
34	-	-	-	-	bundle add message	bundle add message	da controladora para o *switch*
35	-	-	-	-	-	controller status	assíncrona

2.2 Redes Sem Fio Definidas Por Software

As Redes Sem Fio Definidas Por Software têm sido um tema de diversas pesquisas relacionadas à SDN, conforme analisaremos algumas delas, no decorrer desta Seção. Isso se motiva pelo fato do protocolo Openflow ainda não prover suporte às redes sem fio. Vale destacar que o Openflow está continuamente sendo aperfeiçoado e adicionando novas funcionalidades a cada versão. A partir da versão 1.5, por exemplo, ele já é capaz de fazer um tratamento específico para os fluxos que entram e saiam por uma mesma porta, comportamento típico para as interfaces dos dispositivos das redes sem fio(OPENNETWORKING.ORG, 2014b(. Apesar destes avanços, ainda não contempla as informações e demais características relacionados às diversas tecnologias de redes sem fio.

[16] assíncrona - são geradas pelo *switch* para atualizar o controlador sobre eventos da rede e mudanças no estado do *switch*.

2.2.1 ODIN e OpenSDWN

ODIN é um framework SDN, proposto por Suresh (SURESH et al., 2012, que busca simplificar a implementação de serviços de alto nível em WLAN empresariais, através de um conjunto de abstrações e interfaces de diferentes serviços WLAN que podem ser implementados como aplicações de rede. O ODIN utiliza a troca de informações entre módulos instalados na controladora e no cliente. ODIN é baseado no paradigma SDN, no qual a WLAN é orquestrada através de um controle central gerido pelo conjunto de aplicações de rede. Dentre eles, podemos destacar a AAA (Authentication, Authorization and Accounting)[17].

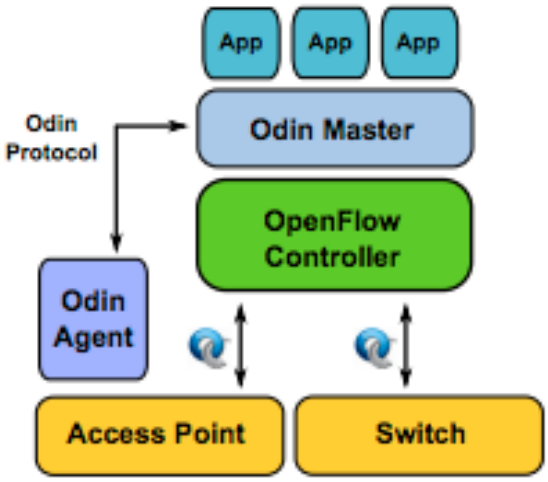

Figura 5 – Ilustra a topologia e comunicação da arquitetura ODINSURESH et al., 2012

O ODIN necessita de módulos complementares, que são: o agente ODIN em cada ponto de acesso e o coordenador centralizado (*ODIN Master*) que adapta a funcionalidade da controladora Openflow. A topologia do Openflow proposta pelo ODIN, tenta suprir a falta de suporte nativo do Openflow, como pode ser ilustrado na Figura 5. Essa abordagem demanda recursos adicionais computacionais e gera outro fluxo de gerência e dados, à parte do Openflow.

O ODIN também trata de alguns desafios especícos em WLANs, como os clientes IEEE 802.11 associarem-se a um ponto de acesso, o que ocorre com decisões baseadas inteiramente em escolhas locais. Para este fim, o ODIN constrói um ponto de acesso virtual que controla o processo de associação, e simplifica a gestão na aplicação cliente. Com os pontos de acesso virtuais, o usuário tem a impressão que está conectado ao seu ponto de acesso, em toda a área de cobertura.

[17] AAA (Authentication, Authorization and Accounting) é uma referência aos protocolos relacionados com os procedimentos de autenticação, autorização e auditoria. Em segurança da informação, a autenticação é responsável por verificar a identidade digital do usuário de um sistema, a autorização é o mecanismo baseada em restrições que garante que um usuário autenticado somente tenha acesso aos recursos aos quais possui autorização e, a auditoria refere-se a coleta de informações sobre o uso dos recursos de um sistema e pode ser utilizada para gerenciamento, planejamento ou cobrança.

O OPENSDWN estende o ODIN com a programação do plano de dados do WIFI (WIFI datapath programmability - WDTX), uma abstração unificada dos pontos de acesso, interfaces participantes e as middleboxes virtualizadas.

Neste trabalho, é possível, através de agentes e processos que devem residir na controladora, ter acesso às informações sem fio e, através do protocolo do Odin, contornar a limitação do protocolo Openflow. Não permite que a controladora tenha a sua disposição nenhuma das características desta rede para a sua tomada de decisão, e não pode ser utilizado nenhum emulador ou simulador SDN, para validar este trabalho. A gestão das aplicações e da rede sem fio é externa ao Openflow, demandando um volume adicional de recursos computacionais, o que pode limitar a implementação em alguns sistemas embarcados.

2.2.2 wmSDN

Outra abordagem em aplicação de SDN em redes sem fio é o Wireless Mesh Software Defined Network - WMSDN [18](DETTI et al., 2013) que são redes de rádios com multi-saltos, nas quais cada nó é um roteador IP com uma ou várias interfaces sem fio, normalmente baseadas no IEEE 802.11 WIFI. Essas interfaces WIFI são configuradas no modo ad-hoc, utilizando antenas omnidirecionais. Entretanto, alguns roteadores podem ser configurados como ponto de acesso (Access Point - AP) e, os outros, como estações (STAtion - STA).

O wmSDN trata as redes WMN e foi concebido como um conjunto de ferramentas compostas por um *switch* Openflow (Open v*switch*), controladora POX, daemon OLSR[19] e um conjunto de scripts em BASH[20] e PYTHON. Foi testado no ambiente emulado baseado em Containers[21] LINUX, NS-3 e ferramentas do simulador de rede CORE[22].

Neste trabalho, é possível simular as redes sem fio e controlá-las utilizando uma controladora Openflow, mas a controladora não tem à sua disposição nenhuma das características desta rede, na tomada de decisão pela controladora. Utiliza o suporte nativo do NS-3 ao Openflow, que ainda é bem limitado e desatualizado em relação às novas funcionalidades e versões do Openflow. Outro fator restritivo é que, por utilizar esse suporte do NS-3, a controladora utilizada só pode ser o POX, sendo a controladora compilada embarcada no simulador, ou seja, não é externa ao ambiente e, também, é bem limitada

[18] wmSDN se encontra disponibilizado <https://github.com/netgroup/wmSDN>.

[19] OLSR é um protocolo do roteamento para redes de dados sem fio móveis do tipo Ad-hoc.

[20] BASH é o interpretador de comandos em modo texto, conhecido como shell. Ele é o interpretador padrão da grande maioria das distribuições LINUX.

[21] Containers também são conhecidos como LinuX Containers (LXC). Oferece melhor desempenho, maior segurança, agilidade e flexibilidade na criação e implementação de aplicações.

[22] Common Open Research Emulator (CORE) é um poderoso emulador de redes de computadores, sendo um fork de outro emulador, chamado IMUNES, adicionando a capacidade de emular redes sem fio. Se encontra disponibilizado em <http://www.nrl.navy.mil/itd/ncs/products/core>.

em relação ao suporte a uma versão antiga do Openflow.

2.2.3 MININET-WIFI

O MININET-WIFI[23](FONTES et al., 2015) é um emulador de redes que permite a aplicação dos conceitos de SDN em redes sem fio, também chamado de Software Defined Wireless Networks (SDWN). O MININET-WIFI permite emular diferentes tipos de cenários de redes sem fio como redes infra-estruturadas, ad hoc, mesh e também a mobilidade dos nós. Isso é possível devido a ser um *fork* do projeto do MININET[24], apenas acrescentando novas funcionalidades sem fio. As estações sem fio e os pontos de acesso são virtualizados e o emulador permite criar também os demais dispositivos tradicionais suportados pelo MININET como computadores, comutadores e controladores Openflow.

A emulação para redes sem fio é bastante desafiadora devido a diversos fatores, e podemos destacar interferências[25], seleção do canal, mudanças de acordo com a mobilidade, além de outras características a autenticação e a autorização. É necessário que se utilizem modelos que representem com fidelidade essas características, como largura de banda, atraso, latência e perda de pacotes, conforme ocorreria em um ambiente real. Para emular essas características, o utilitário *LINUX TC (Traffic Control)* é empregado para fazer o controle das interfaces virtuais, podendo customizar os valores para largura de banda, atraso, latência e perda de pacotes, a fim de representar de forma fidedigna o comportamento desta troca de dados.

A principal forma realística para experimentar WIFI e Openflow é através de pontos de acesso com código-fonte aberto, como o OpenWRT[26] , que permite transformar simples roteadores sem fio em comutadores Openflow. O OpenWRT possui limitações de recursos, escalabilidade e controle. O MININET-WIFI, se propõe a trabalhar nestes tipos de adversidades, ao possuir rápida prototipação e razoável escalabilidade, além dos demais benefícios já conhecidos pela extensa comunidade do MININET. O MININET-

[23] MININET-WIFI é uma ramificação do projeto MININET que apenas adiciona estações e pontos de acesso WIFI, mantendo as demais funcionalidades do MININET e se encontra disponibilizado em <https://github.com/intrig-unicamp/mininet-wifi>.

[24] MININET é um emulador de redes, desenvolvido por pesquisadores da Universidade de Stanford nos Estados Unidos, com o objetivo de apoiar pesquisas colaborativas permitindo protótipos autônomos de redes definidas por software, para que qualquer pessoa possa fazer o download, executar, avaliar, explorar e ajustar. Ele é capaz de emular links, hosts, switches e controladores, utilizando processos que rodem em espaços de nomes da rede (network namespaces) e pares Ethernet virtuais.

[25] Principais Fontes de interferência na faixa de 2,4 GHz ou 5 GHz: Micro-ondas; Serviços Diretos de Satélite (DSS); Determinadas fontes elétricas externas, como linhas de alimentação, trilhos de ferrovias elétricas e estações de energia; Telefones sem fio; Transmissores de vídeo (transmissores/receptores); Alto-falantes sem fio ; Alguns monitores externos e telas LCD: algumas telas podem emitir interferência harmônica, principalmente na banda de 2,4 GHz entre os canais 11 e 14; Qualquer outro dispositivo sem fio que opere na largura de banda de 2,4 GHz ou 5 GHz (câmeras, babás eletrônicas, dispositivos sem fio de vizinhos, dentre outros.). Além disso, alguns materiais geram interferências dentre eles podemos detacar os metais, vidro, concreto, gesso.

[26] OpenWRT pode ser caracterizado como uma distribuição Linux para dispositivos embarcados e se encontra disponibilizado em <https://openwrt.org/>.

WIFI estendeu alguns recursos do MININET para suportar alguns modelos de mobilidade de usuários sem fio, modelos de propagação do sinal (Friis Propagation Loss Model[27] e o Two-Ray Ground Propagation Loss Model[28]), além de permitir a possibilidade de roteamento personalizado para redes mesh.

Existem alguns fatores que restringem o emulador, como o número máximo de 100 dispositivos sem fio (AP e estações), por limitação do módulo responsável pela criação das placas de redes virtuais (pode ser recompilado para ultrapassar esse limite), além da não representação do meio físico de forma adequada como por exemplo a propagação no meio que não responde, com exatidão, às interferências de sinal e com as taxas de transferência de dados coerente ao encontrado em um ambiente real. Tais limitações deverão ser tratadas em futuras versões e, portanto, são consideradas trabalhos futuros.

Este *fork* do MININET é um ambiente bastante promissor e provavelmente em breve terá o suporte necessário para que seja utilizado em futuras implementações do nosso *framework*. Infelizmente, apesar de ter avançado bastante em relação às suas primeiras versões até a conclusão deste trabalho, ainda não possui as características necessárias de Enlace e Física. Neste trabalho, é possível simular as redes sem fio e controlá-las utilizando uma controladora Openflow; já que a controladora não tem à sua disposição nenhuma das características desta rede, na tomada de decisão pela controladora.

2.2.4 OpenNet

O OpenNet[29] se define como um simulador de redes que foi construído sobre o MININET e o NS-3, a fim de viabilizar o estudo de SDWLAN (Software-Defined Wireless Local Area Network), cuja principal contribuição é um ajuste na implementação do comportamento de varredura do canal sem fio pela estação(CHAN et al., 2014).

Este projeto tomou como base uma característica do NS-3 de poder trabalhar no modo de emulação ou, também conhecido, como de tempo real. Vale destacar que tal característica é detalhada na página do projeto do MININET, como ferramenta de integração entre eles. Neste modo de tempo real, o simulador NS-3 pode trocar pacotes com o mundo real (fora do ambiente do NS-3), o que permite que pacotes originados dentro de nós simulados possam ser processados pela rede real, ou inserir pacotes reais dentro da rede simulada. Essas duas abordagens são permitidas através dos módulos FdNetDevice e TapBridge do simulador NS-3.

[27] Friis Propagation Loss Model é um modelo de propagação que só é válido para a propagação no espaço livre dentro da chamada região de campo distante (maior que 3 vezes o comprimento de onda).

[28] Two-Ray Ground Propagation Loss Model é um modelo de propagação de rádio que prevê a perda no percurso quando o sinal recebido consiste da linha de visada e acrescido de um multi-caminho formado predominantemente por uma única onda refletida.

[29] OpenNet se encontra disponibilizado em <https://github.com/dlinknctu/OpenNet>.

O módulo FdNetDevice permite ler e escrever em um descritor de arquivos que fica associado com o dispositivo de rede (através de um *socket raw*), o que permite mesclar o NS-3 com um ambiente real e, com isso, gerar experimentos com resultados replicáveis em um ambiente de redes do mundo real e, ainda, permitir utilizar as ferramentas de coleta e tratamento estatístico do NS-3.

O módulo TapBridge permite que um host real participe de uma simulação do NS-3, como se este fosse um nó simulado; neste cenário o NS-3 conecta com uma interface virtual TAP criada no host LINUX. Os pacotes enviados pelo host para o dispositivo TAP são transmitidos através de um descritor de arquivos para o processo do NS-3. Após isto, é encaminhado para o TapBridge, que passa para o dispositivo de rede no contexto do NS-3 e, por fim, é transmitido pelo canal emulado no NS-3. Uma aplicação, para essa implementação, é o uso de aplicações nativas de dispositivos externos ao NS-3 (como é o caso do Openflow) além de permitir a sua análise no NS-3.

Para isso, é necessária a utilização de dois dispositivos de rede com TapBridge, para criar um link emulado entre as duas interfaces virtuais TAP. Para essa comunicação, é necessário a criação de dois nós fantasmas dentro do NS-3, no qual cada nó consiste de um TapBridge e um dispositivo de rede NS-3. Esses dispositivos se conectam através do canal emulado pelo NS-3, pelo qual as interfaces TAP funcionam como dispositivos terminais do *link* emulado.

Neste trabalho, que se encontra em constante atualização no repositório do Github, ainda possui problemas no processo de conectividade entre os nós ao se continuar a mobilidade e ser necessário um segundo processo de associação do nó móvel. Apesar de ser possível simular as redes sem fio e controlá-las utilizando uma controladora Openflow, a controladora não tem a sua disposição nenhuma das características desta rede na tomada de decisão pela controladora.

2.2.5 Aetherflow

Durante a fase de coleta de resultados do nosso trabalho de pesquisa, ao continuar a análise do estado da arte, também se buscou estender o modelo genérico do SDN, para suportar explicitamente interfaces sem fio e pontos de acesso.

O AetherFlow(YAN et al., 2015) utilizou também as mensagens *Experimenter* para transportar as mensagens que o mesmo definiu, que são: mensagens de notificação à controladora; pedido e resposta de estatísticas de porta lógica; pedido de configuração de porta física; pedido de configuração de porta lógica; pedido e resposta de capacidade da porta física; forçar desassociação de uma estação e mensagem de erro.

A implementação foi em um TP-LINK WR1034ND v2 que possui cinco interfaces de 100Mbps Ethernet e uma interface de rádio, que suporta o IEEE 802.11b/g/n, com

o *firmware* do OpenWRT 14.07 Barrier Breaker. Como o OpenWRT não suporta nativamente o SDN, também utilizou o CPqD Soft*switch* (ofsoft*switch*1.3). Quando um evento, relacionado com uma estação móvel, é disparado, um sumário é enviado por um processo residente ao ofsoft*switch*, que por sua vez o encaminha para a controladora utilizando uma mensagem de evento de porta.

Este trabalho é um *testbed* que utiliza uma implementação de um *framework* para estender o Openflow, através da mensagem *experimenter*, o que está alinhado com o objetivo da presente pesquisa. Não foi utilizado o próprio *switch* Openflow para fazer o recebimento das características sem fio, sendo necessário um processo adicional no ponto de acesso, como é necessário no ODIN. Apesar de obter as informações da rede sem fio, no teste apresentado no artigo foi arbitrado um tempo específico, não utilizando o conteúdo das mensagen, enviadas através do *framework*, com as características desta rede para a tomada de decisão pela controladora.

2.2.6 Consideracoes Finais

Conforme destacado no final de cada sub-seção, a maioria dos trabalhos relacionados a Redes Sem Fio Definidas Por Software, com a exceção do WMSDN que trata de redes Mesh, abordam redes sem fio Locais (Wireless Local Area Networks - WLAN) e nem todos são capazes de lidar com outros tipos de redes. Portanto, não suportam redes heterogêneas. A grande maioria dos trabalhos não utiliza nenhuma das informações do meio sem fio para que a controladora tome a decisão e, os que o fazem, não podem ser avaliados em um ambiente emulado ou simulado.

Segundo (HAQUE; ABU-GHAZALEH, 2016), a pesquisa de redes sem fio, utilizando SDN, pode ser aplicada em quatro principais classes de redes sem fio: Wireless Cellular Networks (WCNs), Wireless Sensor Networks (WSNs), Wireless Mesh Networks (WMNs) e Wireless Home Networks (WHNs). Também apresenta a utilizaçção de SDN em outros grupos de redes como a WAN, ad-hoc, veiculares, Device-to-Device (D2D), contexto social e *smart grids communications networks*. Neste trabalho de *survey* são apresentados a deficiência de metodologias de experimentação e avaliação de desempenho, pois a maioria das propostas dos modelos não foram avaliados apropriamente e não disponibilizaram os seus dados. Consideramos que o desenvolvimento de ferramentas de experimentação ou de simulação são fundamentais para melhorar a compreensão destas soluções. Tal entendimento permitirá aumentar o ritmo de inovação na área.

Podemos constatar, pelo que já foi apresentado neste capítulo, que existe a necessidade de mais trabalhos sobre o tema de Redes Sem Fio Definidas Por *Software*, especialmente em relação à integração da gestão das redes sem fio com o protocolo OpenFlow. Nenhum dos trabalhos relacionados modificou apenas os componentes do sistema OpenFlow para obter e enviar as características sem fio. O nosso /textitframework vem

disponibilizar um meio de comunicação flexível e customizável, que permite inovar com o protocolo OpenFlow, acrescentando a este as funcionalidades que inexistem em sua especificação e em trabalhos correlacionados.

3 Proposta de Solução

Este Capítulo tem como objetivo apresentar ao leitor o FREESDWN (*Framework for Realistic Experiments using Software Defined Wireless Networks*). Desta forma, a Seção 3.1 caracteriza o FRESDWN e na Seção 3.2 é apresentada a implementação do FRESDWN.

3.1 Definição do FRESDWN

O FRESDWN permite o desenvolvimento de *software* de controle personalizado, possibilitando que sejam utilizados serviços a fim de atender às necessidades dos usuários através da extensão do protocolo Openflow. A troca de mensagens *Experimenter* foi o mecanismo empregado para permitir essa inovação e é utilizada a fim de suportar os princípios de rede sem fio. Essas mensagens são simétricas, e portanto, podem ser geradas tanto pela controladora quanto pelo *switch*. Essas mensagens são enviadas sem solicitação prévia. Nas abordagens SDN típicas, até o momento, incluindo o Openflow, não se dispõe de mecanismos capazes de lidar com as complexidades do ambiente de rede sem fio. Portanto, devem ser buscadas novas maneiras de se pensar o SDN para tratar essas complexidades inerentes às redes sem fio.

O FRESDWN é flexível o suficiente para permitir o envio de mensagens sem confirmação e mensagens com confirmação. As mensagens sem confirmação, ilustradas na Figura 6 e na Figura 7, podem ser definidas como aquelas que suportam o envio de uma única mensagem Experimenter pelo controlador ou pelo *switch*. As mensagens com confirmação, ilustradas na Figura 8 e na Figura 9, podem ser definidas como aquelas que suportam o envio de uma única mensagem Experimenter pelo controlador ou pelo *switch*, mas que esperam uma resposta da solicitação pela sua contraparte.

A Figura 6 ilustra um ambiente aonde tanto a controladora quanto o *switch* possuem suporte ao FRESDWN e que a mensagem do FRESDWN foi iniciada pela controladora com base no algoritmo nela implementado. Com base nesta solicitação de informação da controladora, o *switch* Openflow busca a informação. Esse tipo de mensagem também pode ser utilizada para enviar um comando que modifique características do *switch* já que ela pode ser generalizada de acordo com a implementação do FRESDWN. Consta também que todas as funcionalidades complementares do plano de controle suportadas pelo protocolo o Openflow continuam sendo trocadas, independente de quem as tenha iniciado.

A Figura 7 ilustra um ambiente aonde tanto a controladora quanto o *switch* pos-

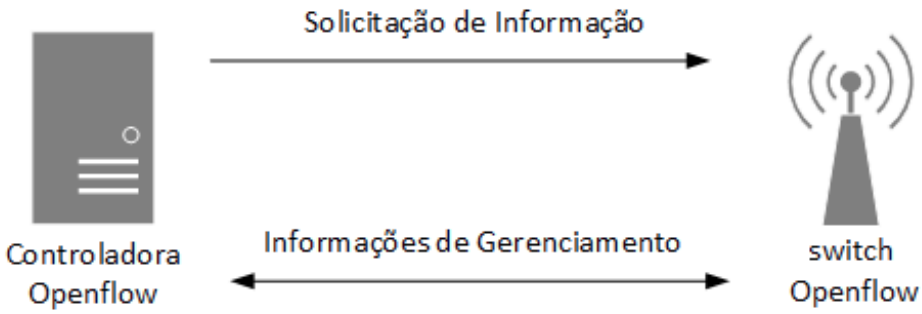

Figura 6 – Ilustra a troca de mensagens Experimenter sem confirmação entre a controladora e o nó sem fio móvel (*switch* Openflow), quando a mensagem do FRESDWN foi iniciada pela controladora

suem suporte ao FRESDWN e que a mensagem do FRESDWN foi iniciada pelo *switch* Openflow com base nas mudanças das informações que as informa para a controladora. Com base nestas informação a controladora analisa, de acordo com o seu algoritmo, e pode apenas registrar a nova informação ou enviar uma mensagem de controle padrão do Openflow para o *switch*. Esse tipo de mensagem também pode ser utilizada para enviar um status do *switch* ou do fluxo de informações por ele tratados para a controladora, já que ela pode ser generalizada de acordo com a implementação do FRESDWN. Relembrando que todas as funcionalidades complementares do plano de controle suportadas pelo protocolo o Openflow continuam sendo trocadas, independente de quem as tenha iniciado.

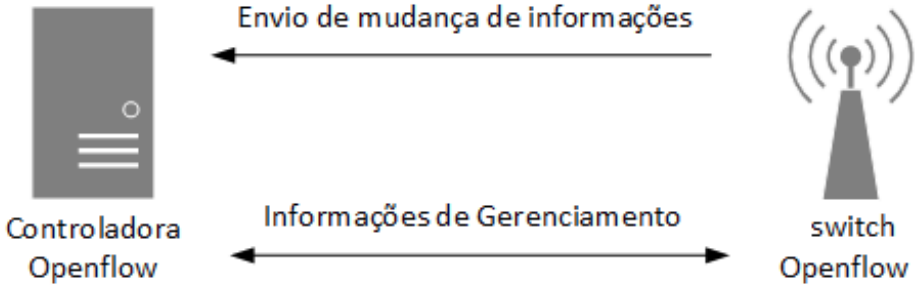

Figura 7 – Ilustra a troca de mensagens Experimenter sem confirmação entre a controladora e o nó sem fio móvel, quando a mensagem do FRESDWN foi iniciada pelo nó movel (*switch* openflow)

A Figura 8 ilustra um ambiente aonde tanto a controladora quanto o *switch* possuem suporte ao FRESDWN e que a mensagem do FRESDWN foi iniciada pela controladora com base no algoritmo nela implementado. Com base nesta solicitação da controladora, o *switch* Openflow trata essa mensagem e retorna uma resposta, também através de uma mensagem FRESDWN. Consta também que todas as funcionalidades complementares do plano de controle suportadas pelo protocolo Openflow continuam sendo trocadas, independente de quem as tenha iniciado.

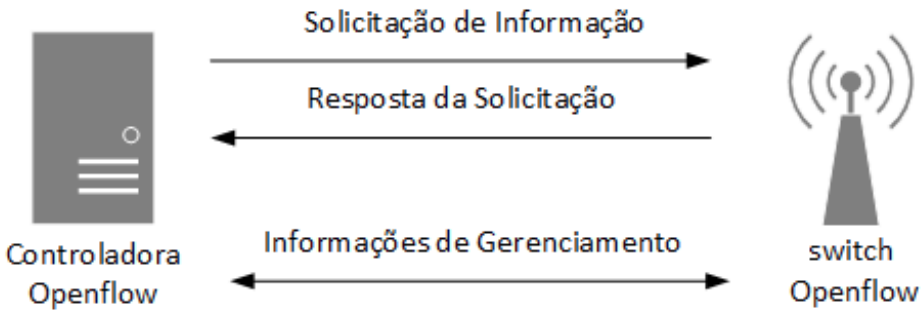

Figura 8 – Ilustra a troca de mensagens Experimenter com confirmação entre a controladora e o nó sem fio móvel (*switch* openflow), quando a mensagem do FRESDWN foi iniciada pela controladora

A Figura 9 ilustra um ambiente aonde tanto a controladora quanto o *switch* possuem suporte ao FRESDWN e que a mensagem do FRESDWN foi iniciada pelo *switch* Openflow, com base nas informações que deseja informar, as mudanças para a controladora. Com base nestas informações a controladora, através de seu algoritmo envia para o *switch* Openflow uma resposta também através de uma mensagem FRESDWN. Essa resposta da controladora pode ser uma simples confirmação ou o início de uma nova solicitação conforme visto na Figura 8. Relembrando que todas as funcionalidades complementares do plano de controle suportadas pelo protocolo Openflow continuam sendo trocadas, independente de quem as tenha iniciado.

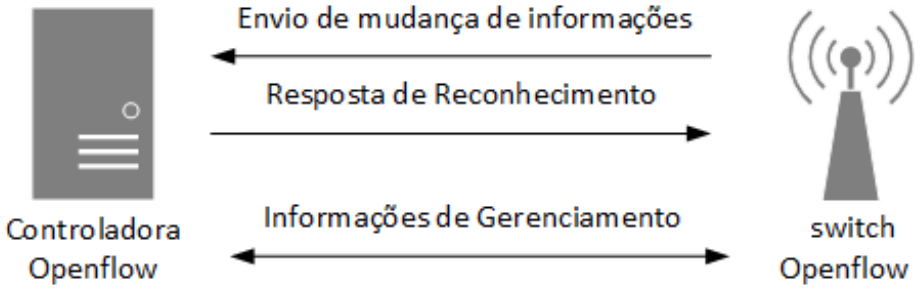

Figura 9 – Ilustra a troca de mensagens Experimenter com confirmação entre a controladora e o nós sem fio móvel, quando a mensagem do FRESDWN foi iniciada pelo nó móvel (*switch* openlflow)

O projeto do FRESDWN, com base na troca de qualquer tipo de informação nas mensagens entre a controladora e os pontos de acesso, não fica restrito a uma única tecnologia de rede sem fio ou uma implementação particular de SDN. O FRESDWN pode ser aplicado a diversas implementações, podendo potencialmente ser aplicado fora do escopo de redes sem fio ao enviar características complementares às dos protocolos atuais dos nós ou ativos de rede para que a controladora tome uma melhor decisão, como deve ser feito em Internet das Coisas. O FRESDWN é capaz de enviar mensagens *Experimenter* que podem transportar informações particulares de nós sem fio, como a mobilidade dos

usuários, consumo de energia dos dispositivos e outras características de redes sem fio, permitindo trocar essas informações entre a rede sem fio e a controladora para a tomada de decisão do plano de controle dentro das mensagens Openflow a fim de permitir o gerenciamento do uso dos recursos da rede sem fio.

3.2 Implementação do FRESDWN

A implementação do FRESDWN é dividida em três componentes básicos: a controladora Openflow (RYU), o *switch* Openflow (ofsoft*switch*13) e o ambiente integrado MININET/NS-3. O ambiente integrado MININET/NS-3 utiliza os *switches* Openflow em cada nó emulado para interagir com a controladora Openflow em um ambiente emulado realístico de SDN. Esse ambiente emulado viabiliza a validação de protocolos, provas de conceitos em um ambiente controlado, ou seja, permite a variação de parâmetros relevantes e a manutenção dos demais constantes, o que permite obter conclusões sobre o impacto destes parâmetros. Outro ponto relevante é a facilidade de avaliar as soluções ou protocolos em múltiplos cenários rapidamente e com um custo reduzido.

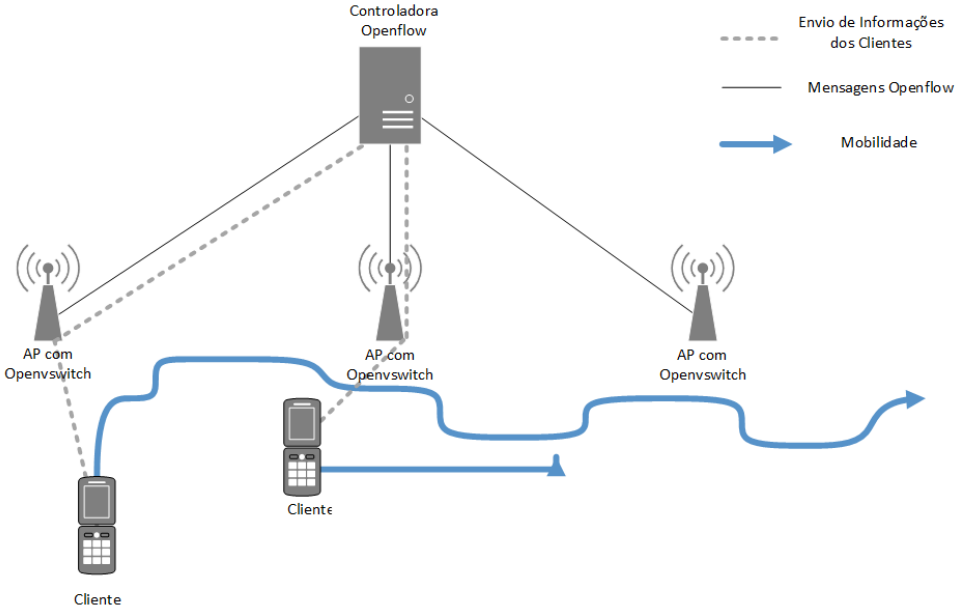

Figura 10 – Ilustra a topologia e comunicação entre os nós sem fio móvel e a controladora

O Openflow, até o momento, assume um canal seguro e confiável entre a controladora e os nós controlados, o que não pode ser garantido em ambientes sem fio, principalmente quando existe mobilidade dos nós ou a possibilidade de obstáculos móveis que impeçam ou comprometam as transmissões. Além desses aspectos, a rede sem fio está sujeita a mudanças nas características do canal sem fio ocasionadas por alterações na atmosfera terrestre ou pela ocorrência de outras fontes de interferência eletromagnética. A Figura 10 ilustra um cenário de mobilidade dos nós, que foi utilizado como validação

do Opennet (CHAN et al., 2014) e permite comparativamente analisar o impacto da mobilidade dos nós na troca de mensagens do FRESDWN com um ambiente, sem o uso do FRESDWN.

Inicialmente foi feito um estudo e ensaios de implementação utilizando o OVS (OpenV*switch*), que é uma implementação de um *switch* Openflow executada à nível de Kernel do Linux. No andamento da pesquisa ela foi descartada, porque possuía fraca documentação e difícil customização, o que inviabilizaria a construção do FRESDWN no tempo hábil, além de ser complexa e exigir retrabalho a sua customização em ambiente multi-plataforma.

Ofsoft*switch*13 é um *switch* Openflow a nível de biblioteca de usuário, que além de possuir uma boa documentação e código modular com suporte ao Openflow 1.3, foi escolhido para a implementação no FRESDWN. Também esse *switch* pode ser empregado sem retrabalho em *testbeds* (*access points* e nós) com outras plataformas e arquiteturas. Isto se deve ao fato de permitir a utilização do mesmo código fonte nesses diversos cenários de utilização.

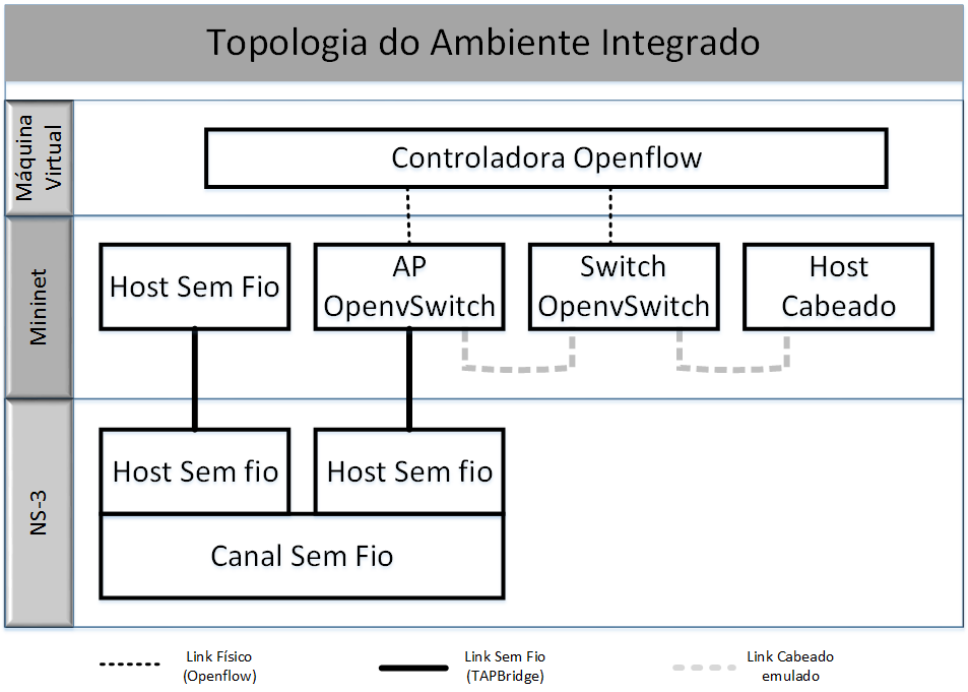

Figura 11 – Detalhamento da Topologia entre Controladora externa, MININET e NS-3

Como podemos observar na Figura 11, o emulador de redes MININET é o responsável pela troca de mensagens entre a controladora externa e os nós de rede sem fio através do protocolo Openflow. Já o simulador de redes NS-3, que está operando no modo emulado, se integra com os dispositivos do MININET e é o responsável pelas características do meio físico sem fio e pela mobilidade dos nós. Portanto, as camadas Física e Enlace, bem como algoritmos para representar a perda do canal e mobilidade dos usuários,

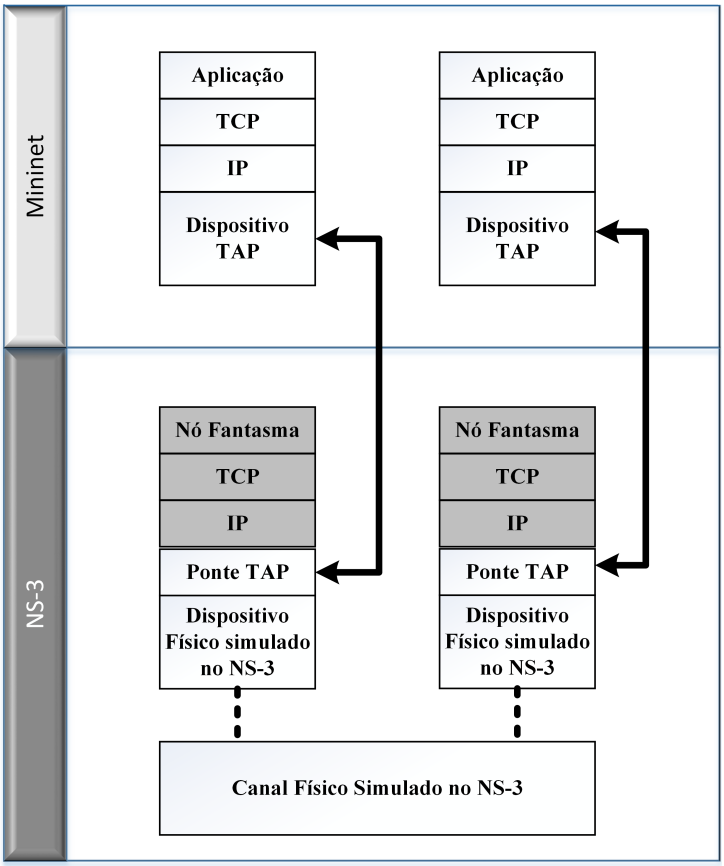

Figura 12 – Detalhamento da Ponte TAP da Topologia entre MININET e NS-3

são processadas no ambiente do NS-3 e a comunicação com as camadas superiores são tratadas pelo MININET.

A comunicação entre as instâncias de cada um dos nós implementados no NS-3 com os nós respectivos no MININET ocorre através das pontes TAP, que são interfaces de redes Ethernet virtuais, a nível de Kernel do Linux, que permitem tunelamento entre os aplicativos, conforme ilustrado na Figura 12. Pode-se observar que no escopo do NS-3 as camadas de rede e superiores não são tratadas no ambiente do NS-3 e, sim, redirecionados aos nós do MININET, da mesma forma que o MININET não trata as informações das camadas de Enlace e Física.

Na primeira versão do Openflow (Openflow 1.0) foi disponibilizada uma mensagem cuja finalidade se encontrava em aberto. Essa troca de mensagens simétricas foi o mecanismo recomendado pela especificação do protocolo Openflow para que os fabricantes fossem capazes de gerar uma inovação customizada para as suas necessidades específicas de maneira flexível. Os pesquisadores também utilizaram essa funcionalidade para implementar, tanto na controladora quanto no *switch*, funcionalidades pelos pesquisadores idealizadas, pois o protocolo preconizava que era através das mensagens do tipo *Vendor* que deveria ocorrer a inovação, conforme esquema ilustrado na Figura 13. As extensões da Nicira, implementadas durante o doutorado em Stanford de seu fundador, foram as

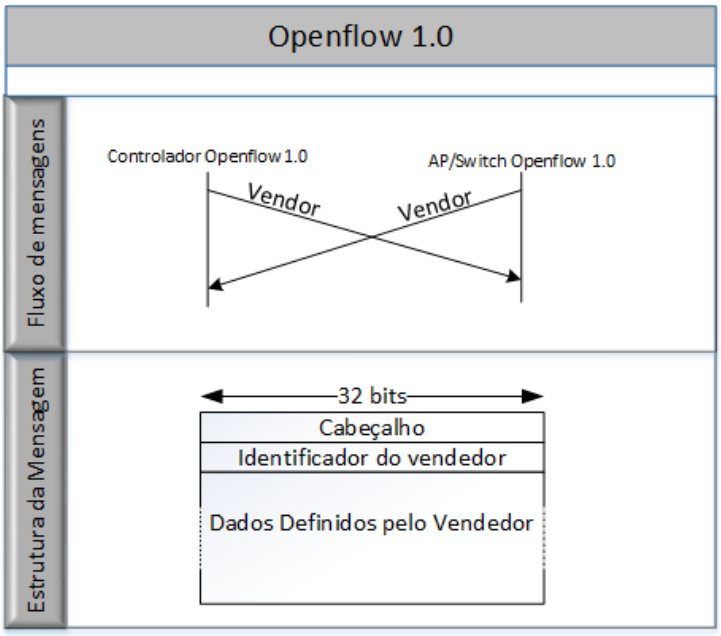

Figura 13 – as especificação da mensagem vendor do Openflow 1.0, adaptada do flowgrammable.

primeiras a utilizar a mensagem vendor para inovar e propor extensões ao Openflow.

A partir do Openflow 1.1, as mensagens simétricas, que permitem a inovação customizada para os fabricantes e também para os pesquisadores, deixam de se chamar Vendor e passam a ser *Experimenter*. Conforme esquema ilustrado na Figura 14, observamos que a partir da versão 1.2 o formato se manteve e permite a customização de sub-tipos de experimentos para o mesmo número de identificador do Experimento (no código do Openflow implementado nos *switch*es e nas controladoras, mesmo nas versões mais recentes, continua sendo chamada esse tipo de Vendor com o intuito de se compatibilizar com a versão 1.0).

```
#define FRESDWN_OUI_STR "0x00000005"
#define FRESDWN_VENDOR_ID 0x00000005
```

Conforme pode ser constatado no trecho de código acima, foi utilizado como identificador do tipo do VENDOR_ID o código do grupo de trabalho da ONF para redes sem fio, apesar desse grupo de trabalho ainda não ter disponibilizado uma solução para esses tipos de redes.

Na implementação da mensagem do FRESDWN, foi otimizada a capacidade de transmissão, utilizando o tamanho típico de MTU que é de 1500 bytes. Permitindo, após os descontos dos demais cabeçalhos, enviar um vetor de dados *Experimenter* de até 1.408 bytes, sendo 1.392 úteis para qualquer aplicação que utilize o FRESDWN. Essa capaci-

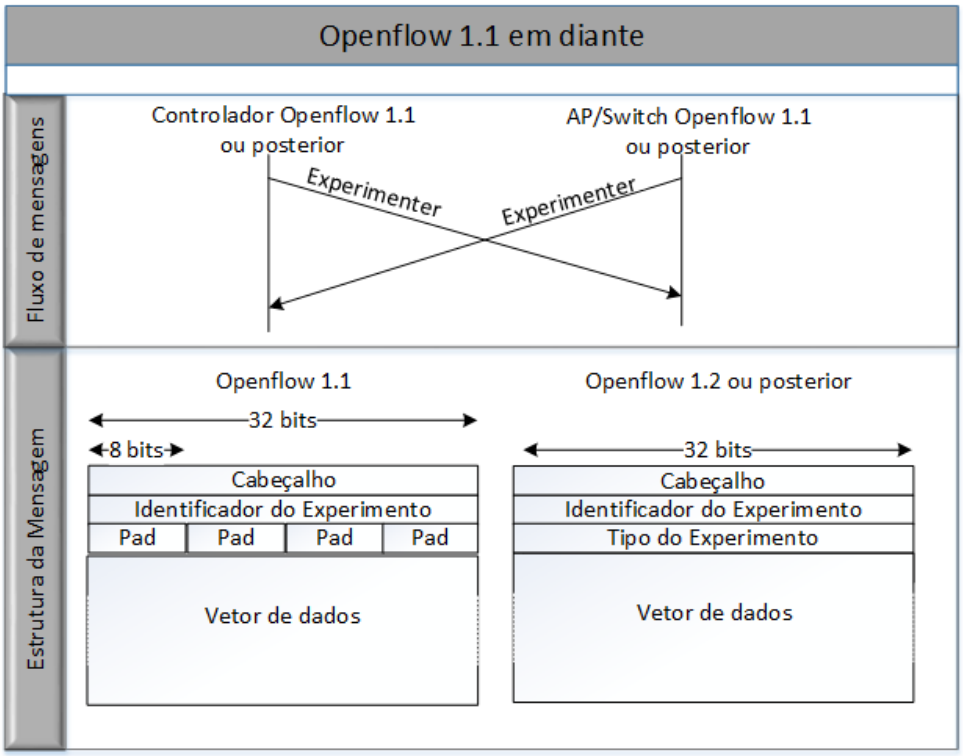

Figura 14 – segundo as especificação do Openflow 1.1 em diante, adaptada do flowgrammable.

dade útil[1] é disponibilizada no campo dummy_content, conforme pode ser constatado na estrutura de dados abaixo.

```
struct fresdwn_dummy {
    struct fresdwn_header header;
    uint8_t dummy_content[FRESDWN_ARRAY_SIZE];
};
OFP_ASSERT(sizeof(struct fresdwn_dummy) == 1408);
```

Esse tamanho de vetor de dados permite que num único pacote de mensagens Openflow sejam enviados um grande volume de informações, desde que a sua codificação seja feita de forma eficiente. O FRESDWN faz a troca de informações da rede sem fio através do JSON (*JavaScript Object Notation* - Notação de Objetos JavaScript), pois é um formato simples e leve para intercâmbio de dados computacionais, especificado na RFC 4627(CROCKFORD, 2006). Uma vantagem adicional nesta notação é sua facilidade de leitura e escrita por seres humanos aliado à sua simplicidade de ser gerado e interpretado pelos componentes computacionais. O JSON é tipicamente usado em ambientes onde há um grande fluxo de dados entre o cliente e o servidor e a otimização é de supra importância. Vem sendo amplamente empregado nas diversas áreas de redes e de sistemas

[1] capacidade útil é a capacidade de carga de dados do FRESDWN, ou seja, o que pode ser transmitido entre a controladora e nó já descontando todos os cabeçalhos.

distribuídos, inclusive em redes sem fio, que necessitam transmitir grande volume de dados rapidamente. Algumas das inovações em que se empregou essa tecnologia para minimizar o uso da rede na troca de dados são: as redes WBAN (*Wireless Body Area Network*), possuem expressivos requisitos de bateria e de velocidade, conforme pode ser ilustrado nos artigos de Yuvaradni (YUVARADNI et al., 2016), alocação de canal em redes sem fio Mesh de Yingxiong (YINGXIONG et al., 2016), em redes sensores que necessitam de escalabilidade (PÖTSCH, 2016), em rede de dispositivos móveis como *smartphones* de Jang (JANG et al., 2016), patente de orquestração de redes de distribuição de videos (WU et al., 2016, dentre outras pesquisas que empregam o JSON.

Com o intuito de reduzir ainda mais o tamanho ocupado pelas informações das características das redes sem fio a serem transmitidas no JSON, foram definidas uma sequência de enumeradores. Os enumeradores são empregados na programação pois têm a vantagem de serem conjuntos de valores que são auto-documentados e permitem serem mais legíveis e menos vulneráveis a erros de programação. Os enumeradores abaixo foram definidos como prova de conceito das informações que podem ser transmitidas e que servirão de base de decisão e gestão pela controladora, conforme podem ser vistos abaixo.

```
/* Freswdn messages types*/
enum fresdwn_type {
    FRESDWNT_FEATURE_REQUEST, // Solicita as caracteristicas do dispositivo
    FRESDWNT_FEATURE_REPLY, // Informa as caracteristicas para a controladora
    FRESDWNT_STATUS_REQUEST,
    FRESDWNT_STATUS_REPLY,

    /* Generic GET/SET Wireless configuration */
    FRESDWNT_ACT_SET_CONFIG,
    FRESDWNT_ACT_GET_CONFIG,
    FRESDWNT_COMMAND_REQUEST,
    FRESDWNT_COMMAND_REPLY,
    FRESDWNT_FLOW_END_CONFIG,
    FRESDWNT_FLOW_END,
    FRESDWNT_MGMT,

    /* GET/SET Channel configuration */
    FRESDWNT_ACT_SET_CHANNEL_CONFIG,
    FRESDWNT_ACT_GET_CHANNEL_CONFIG,

    /* GET PHYSICAL information */
    FRESDWNT_ACT_GET_AP_CONFIG,
    FRESDWNT_ACT_GET_AP_IN_RANGE_INFO,
    FRESDWNT_ACT_GET_BEACON_INFO,
    FRESDWNT_ACT_GET_NOISE_INFO,
    FRESDWNT_ACT_GET_POSITION_INFO,
```

```
    /* GET/SET SSID configuration */
    FRESDWNT_ACT_SET_SSID_CONFIG,
    FRESDWNT_ACT_GET_SSID_CONFIG,

    /* DUMMY Message for tests*/
    FRESDWNT_DUMMY
};

/* Wireless technology types*/
enum fresdwn_type_wireless_tecnology {
    FRESDWNT_802_11_a,
    FRESDWNT_802_11_b,
    FRESDWNT_802_11_g,
    FRESDWNT_802_11_n,
    FRESDWNT_802_11_ac,
    FRESDWNT_3G,
    FRESDWNT_4G,
    FRESDWNT_5G
};

enum fresdwn_type_of_wireless_atribute {
// 802.11
    FRESDWNT_FREQUENCY, // 2,4gH
    FRESDWNT_CHANNEL_SIZE, // 20mhz
    FRESDWNT_CHANNEL_NUMBER, // 6
    FRESDWNT_TRANSMITION_TYPE, // FHSS
    FRESDWNT_TRANSMITION_MIN_POWER, // 3mW
    FRESDWNT_TRANSMITION_POWER, // 30mW
    FRESDWNT_TRANSMITION_MAX_POWER, // 30mW
    FRESDWNT_SSID, // CONVIDADO - 32chars
    FRESDWNT_BSSID, // 8f:....
    FRESDWNT_BSSID_TYPE, // 1-infrastructure, 2-independent 3-any
    FRESDWNT_CRYPTO_TECNOLOGY, // WPA
    FRESDWNT_CRYPTO_TYPE_OF_KEY, // TPIK
    FRESDWNT_CRYPTO_SIZE_OF_KEY, // 256bits
    FRESDWNT_ANTENNA_POWER, // 2dbi
    FRESDWNT_ANTENNA_TYPE, // omnidirecional
    FRESDWNT_ANTENNA_QUANTITIES, // omnidirecional
    FRESDWNT_DATA_RATE_RX, // 11M
    FRESDWNT_DATA_RATE_SUPPORTED_RX, // 1M,11M,54M   126chars
    FRESDWNT_DATA_RATE_TX, // 11M
    FRESDWNT_DATA_RATE_SUPPORTED_TX, // 1M,11M,54M   126chars
    FRESDWNT_MAC_ADDRESS,
    FRESDWNT_MEDIUM_OCCUPANCY_LIMIT, // 0..1000
    FRESDWNT_CONTENTION_FREE_POLLABLE, // TRUE
    FRESDWNT_CONTENTION_FREE_PERIOD, // 0..255
    FRESDWNT_CONTENTION_FREE_MAX_DURATION, //0..65536
```

```
    FRESDWNT_AUTHENTICATION_RESPONSE_TIMEOUT, // 1..4294967295
    FRESDWNT_PRIVACY_OPTION_IMPLEMENTED, // FALSE (WEP DESABILITADO)
    FRESDWNT_POWER_MANAGEMENT_MODE, // 1-active  2-powersave
    FRESDWNT_BEACON_PERIOD, // 1..65535
    FRESDWNT_DTIM_PERIOD, // 1..255
    FRESDWNT_ASSOCIATION_RESPONSE_TIMEOUT, //1..4294967295
    FRESDWNT_DISASSOCIATE_REASON, // 1..65536
    FRESDWNT_DISASSOCIATE_STATION, // MAC
    FRESDWNT_DEAUTHENTICATE_REASON, // 1..65536
    FRESDWNT_DEAUTHENTICATE_STATION, // MAC
    FRESDWNT_AUTHENTICATE_FAIL_STATUS, // 1..65535
    FRESDWNT_AUTHENTICATE_FAIL_STATION, // MAC
    FRESDWNT_AUTHENTICATION_ALGORITH, // 1-OPEN SYSTEM    2-SHARED KEY
};

enum fresdwn_type_of_frequency_atribute {
// 802.11
    FRESDWNT_2_4Ghz,
    FRESDWNT_5Ghz,
// CELULAR
    FRESDWNT_850Mhz,
    FRESDWNT_900Mhz,
    FRESDWNT_1700Mhz,
    FRESDWNT_1800Mhz,
    FRESDWNT_1900Mhz,
    FRESDWNT_2100Mhz,
};

enum fresdwn_action_subtype {
    FRESDWN_ACTION_SEND,
    FRESDWN_ACTION_RECEIVE,
    FRESDWN_ACTION_PROCESSING

};
```

Utilizando os conceitos estabelecidos anteriormente, o FRESDWN define alguns tipos de ações que podem ser feitas pela controladora a cada nó sem fio. Dentre elas podemos destacar mensagens que permitem:

- Modificar / Requisitar / Responder
  - características físicas de redes sem fio;
  - características lógicas de redes sem fio;
- Forçar um *host* sem fio a;

- desassociar do AP a qual está conectado;
- associar a um determinado AP.

Características físicas das redes sem fio, que permitem minimizar ou resolver alguns dos problemas de meios não guiados: dentre essas características podemos destacar a potência do sinal, protocolo utilizado na transmissão de dados, características de mobilidade (por exemplo: velocidade relativa, coordenada GPS), dentre outras inerentes a uma tecnologia sem fio particular. Esses envios permitem uma gestão mais precisa e eficiente por parte da controladora dos nós da rede, sem a necessidade de agentes ou módulos externos como utilizado no ODIN (SURESH et al., 2012). Um exemplo de aplicabilidade destas características pode ser a controladora analisar uma região densamente povoada e decidir efetuar o ajuste da potência de transmissão de um conjunto de nós sem fio, com o objetivo de minimizar a interferência de nós adjacentes.

As características lógicas permitem a associação de conceitos complementares de redes sem fio, como por exemplo: o SSID associado, se pertence a um ESSID, protocolos de autenticação (AAA), técnicas de criptografia utilizadas, bilhetagem (no caso de provedores de acesso), dentre outras características. Um exemplo deste processo pode ser o ajuste da autenticação e número de clientes associados a cada ponto de acesso, de acordo com critérios estabelecidos pela controladora.

Outro componente é a gestão da associação e utilização da rede sem fio pelos nós, como por exemplo, o processo do *handoff* de nós móveis entre pontos de acesso, ou o *handover* entre tecnologias sem fio com base em critérios estabelecidos pela controladora.

Também são definidas ações básicas de transmissão e recebimento de mensagens do FRESDWN, para viabilizar a granularidade de sub-tipos de mensagens FRESDWN, o que permite a ampliação das suas funcionalidades, de acordo com as particularidades do objetivo a ser validado ao empregar o FRESDWN. O FRESDWN é flexível e pode ser estendido de acordo com a sua implementação, pois utiliza o conceito de cabeçalhos adicionais que permitem a subdivisão da mensagem experimenter em um conjunto de outras ramificações das mensagens. Como prova de conceito, foram tratadas na implementação do FRESDWN as estruturas básicas de envio e recebimento de cabeçalho que se encontram abaixo ilustradas.

```
/* Action structure for FRESDWN_ACTION_SEND. */
struct fresdwn_action_send {
    uint16_t type;                  /* OFPAT_VENDOR. */
    uint16_t len;                   /* Length is 16. */
    uint32_t vendor;                /* FRESDWN_VENDOR_ID. */
    uint16_t subtype;               /* FRESDWN_ACTION_SEND. */
    uint16_t in_port;               /* New in_port for checking flow table. */
    uint8_t pad[4];
```

```
};
OFP_ASSERT(sizeof(struct fresdwn_action_send) == 16);

/* Action structure for FRESDWN_ACTION_RECEIVE. */
struct fresdwn_action_receive {
    uint16_t type;                  /* OFPAT_VENDOR. */
    uint16_t len;                   /* Length is 16. */
    uint32_t vendor;                /* FRESDWN_VENDOR_ID. */
    uint16_t subtype;               /* FRESDWN_ACTION_RECEIVE. */
    uint16_t in_port;               /* New in_port for checking flow table. */
    uint8_t pad[4];
};
OFP_ASSERT(sizeof(struct fresdwn_action_receive) == 16);

/* Header for FRESDWN-defined actions. */
struct fresdwn_action_header {
    uint16_t type;                  /* OFPAT_VENDOR. */
    uint16_t len;                   /* Length is 16. */
    uint32_t vendor;                /* FRESDWN_VENDOR_ID. */
    uint16_t subtype;               /* FRESDWN_ACTION_*. */
    uint8_t pad[6];
};
OFP_ASSERT(sizeof(struct fresdwn_action_header) == 16);
```

Uma forma de minimizar o ônus da latência das redes SDN seria optarmos por uma instalação proativa destas regras, ou seja, computar antecipadamente as regras de encaminhamento, em contraponto à instalação reativa que só cria a regra após a chegada do primeiro pacote do fluxo. A abordagem proativa é complexa de ser implementada em redes sem fio, principalmente ao levarmos em conta as características de mobilidade e demais particularidades das tecnologias sem fio.

Dentro desse contexto, o FRESDWN se propõe a disponibilizar informações de redes sem fio, com redução de uso dos recursos computacionais, ao abarcar na extensão do protocolo Openflow as características que precisariam de processos complementares, além de utilizar técnicas que minimizem o consumo de banda como a utilização de JSON e enumeradores. Tais abordagens permitem à controladora uma melhor tomada de decisão com reduzida latência de rede, dependendo do algoritmo utilizado pela controladora, como decisões baseadas em técnicas de inteligência artificial, predição de mobilidade baseado em informações históricas ou contexto social e demais estatísticas históricas de comportamento da rede. Tudo isso para a controladora ter os dados necessários para analisar a rede e elaborar proposições da melhor regra para os fluxos e, por conseguinte, podemos ilustrar com a possibilidade de estabelecer as validações da proatividade nas regras a serem definidas no plano de controle.

4 Avaliação de desempenho de um estudo de caso

Neste capítulo, iremos apresentar um conjunto de avaliações experimentais do 3xtitframework FRESDWN, em um cenário específico de uma rede sem fio em uma área livre entre prédios de um campus universitário. Esta avaliação tem, como intuito, responder às seguintes questões:

- O uso do FRESDWN permite aplicar os conceitos de SDN em redes sem fio?
- As informações adicionais sobre a rede sem fio podem ser utilizadas para uma melhor decisão da controladora?
- Qual é o impacto no plano de controle destas mensagens Experimenter contendo as características sem fio?
- Quais são os requisitos mínimos para o funcionamento do ambiente emulado integrado MININET/NS-3?
- Qual é o comportamento do FRESDWN, ao aumentar a mobilidade dos nós?

O restante deste capítulo está organizado da seguinte maneira. Na Seção 4.1 apresentaremos a contextualização e objetivos do estudo de caso usando o FRESDWN. A descrição do cenário utilizado é abordado na Seção 4.2. As configurações e parâmetros utilizados são detalhados na Seção 4.3.

Para apresentarmos os resultados montamos dois estudos de caso: (a) Análise de um cenário possuindo nós com baixa mobilidade, que será detalhado na Seção 4.4; (b) Análise de um cenário possuindo nós com alta mobilidade, que será descrito na Seção 4.5.

4.1 Contextualização e objetivos do estudo de caso do FRESDWN

Neste estudo de caso foi realizada uma emulação de rede SDN, usando o MININET, versão 2.2.1, que se comunica com a controladora RYU, versão de maio de 2015. Adicionalmente foi utilizado o simulador de redes NS-3, versão 3.22, para representar o comportamento dos nós sem fio, tanto na camada Física, quanto na camada de Enlace, bem como a mobilidade dos nós. Desta forma, a emulação de redes SDN usará características simuladas de um ambiente sem fio e, por isso, diremos que este estudo de caso é uma emulação assistida por simulação. Foi escolhido o envio de mensagens sem confirmação apenas, pois o estudo tem características de tempo real e portanto, pressupõe que não há a

necessidade de reenvio de uma possível mensagem experimenter perdida e que o *overhead* introduzido pela confirmação no envio das mensagens com confirmação do FRESDWN seria desnecessário.

Neste ambiente de emulação integrado, foram sendo executados experimentos para coletar as informações sobre os nós e avaliá-las com base na topologia. Estas informações são trocados entre os pontos de acesso e a controladora, através de mensagens do tipo Experimenter, além das demais informações normais de troca de dados e do plano de controle.

O FRESDWN suporta a troca de mensagens da controladora, solicitando aos nós suas características. Tal mensagem pode ser direcionada para um único nó ou vários nós. Também permite o retorno das informações, encaminhada do nó móvel para a controladora, com o resultado da solicitação.

Na análise dos resultados, para verificar o comportamento da solução, foi comparado o ambiente tradicional SDN com a utilização do FRESDWN, ou seja, sem o envio de mensagens sem confirmação contendo informações específicas sem fio para a controladora e com o envio de mensagens sem confirmação *Experimenter*, contendo tais informações.

O FRESDWN foi avaliado em duas etapas: inicialmente foi verificada a efetividade do sistema em permitir a troca de informações da rede sem fio, através de mensagens *Experimenter* do protocolo Openflow. Posteriormente, foi avaliada a eficácia do sistema, analisando o parâmetro **do número de mensagens *Experimenter* em relação às demais mensagens Openflow**, ao realizar um aumento da quantidade e da mobilidade dos nós.

4.2 Descrição do Cenário

O cenário utilizou 6 AP's e cada estudo de caso terá dois ou três nós que podem ser móveis, cuja mobilidade se encontra restrita à área de cobertura de, pelo menos, um ponto de acesso, conforme a Figura 15.

Todos os 6 APś são habilitados em Openflow estendido com a funcionalidade do FRESDWN, a fim de permitir enviar informações complementares sobre o estado da rede que não são suportadas pelo Openflow. Foi implementada uma extensão do ofsoftswitch13 do CPqD que suporta a troca de mensagens Experimenter. Todos os dispositivos sem fio que desejam enviar mensagens complementares para a controladora devem estar executando esse *switch* Openflow customizado. Cada AP possui várias interfaces Ethernet e uma interface WIFI. Cada AP se conecta a uma controladora Openflow real e aos demais $AP's$ pelas interfaces Ethernet.

No ambiente que utiliza o FRESDWN, cada AP envia uma mensagem Experi-

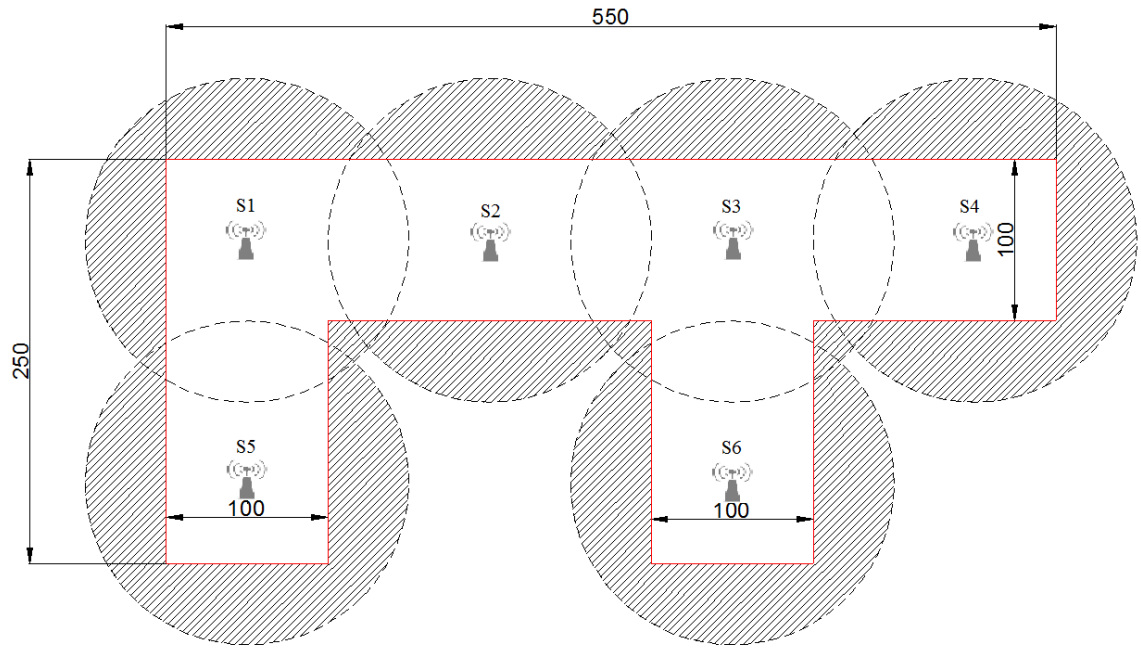

Figura 15 – Disposição dos pontos de acesso e definição da área delimitada de movimentação dos nós nos cenários

menter no momento em que surge um novo nó na topologia, ou que ocorra uma alteração das características sem fio. No ambiente de comparação em que não utiliza o FRESDWN, os $AP's$ atuam como um *switch* L2, que aprende de acordo com a camada de Enlace, e não é enviada nenhuma mensagem Experimenter.

Neste trabalho definimos como:

- Associação: todos os nós que desejam enviar uma informação pela rede sem fio, devem iniciar um processo de associação, no caso de já estarem autenticados na rede. Este serviço mapeia as estações que podem estar associadas a um AP. Uma estação somente pode estar associada a um único AP de cada vez. O AP, por sua vez, pode se associar a várias estações em determinado instante de tempo.

- Desassociação: é o processo de finalizar uma associação existente, devido a uma migração ou desligamento. A desassociação poder ser requisitada tanto pelo nó quanto pelo AP.

Como o ambiente é integrado entre o MININET e o NS-3, a camada Física e de Enlace é feita pelo NS-3, correspondendo aos protocolos 802.11 e o 802.2, além de ser responsável pela mobilidade. A camada Física está sendo tratada pelo NS-3 especificamente pelo módulo YansWifiPhyHelper, que se originou no projeto YANS (Yet Another Network Simulator), que permite uma simulação realística do meio físico, incluindo configurações de perda na propagação. O NS-3 através do TAP envia esse comportamento

realístico para o MININET, responsável pelas demais camadas da pilha de protocolos, que fica responsável por enviar as mensagens Openflow para a controladora externa (RYU).

A fim de observar as características da rede sem fio que podem ser utilizadas para uma melhor tomada de decisão pela controladora SDN, foi implementado na controladora, como prova de conceito do uso do FRESDWN, um simples mecanismo que antecipa o momento de desassociação dos nós. Ao atingir um determinado nível de potencia de sinal, é enviada, pela controladora, uma mensagem para que o ponto de acesso efetue a desassociação do nó. Ao receber a mensagem de desassociação, o nó busca em qual ponto de acesso irá se conectar, de acordo com o nível de sinal naquele instante de tempo, como no momento inicial de uma associação sem fio normal. Esse mecanismo será referenciado nos nossos experimentos como sendo *com o uso do FRESDWN*, e o comportamento padrão de conectividade de camada de enlace (sem o envio da mensagem Experimenter sem confirmação pela controladora) como *sem o uso do FRESDWN*.

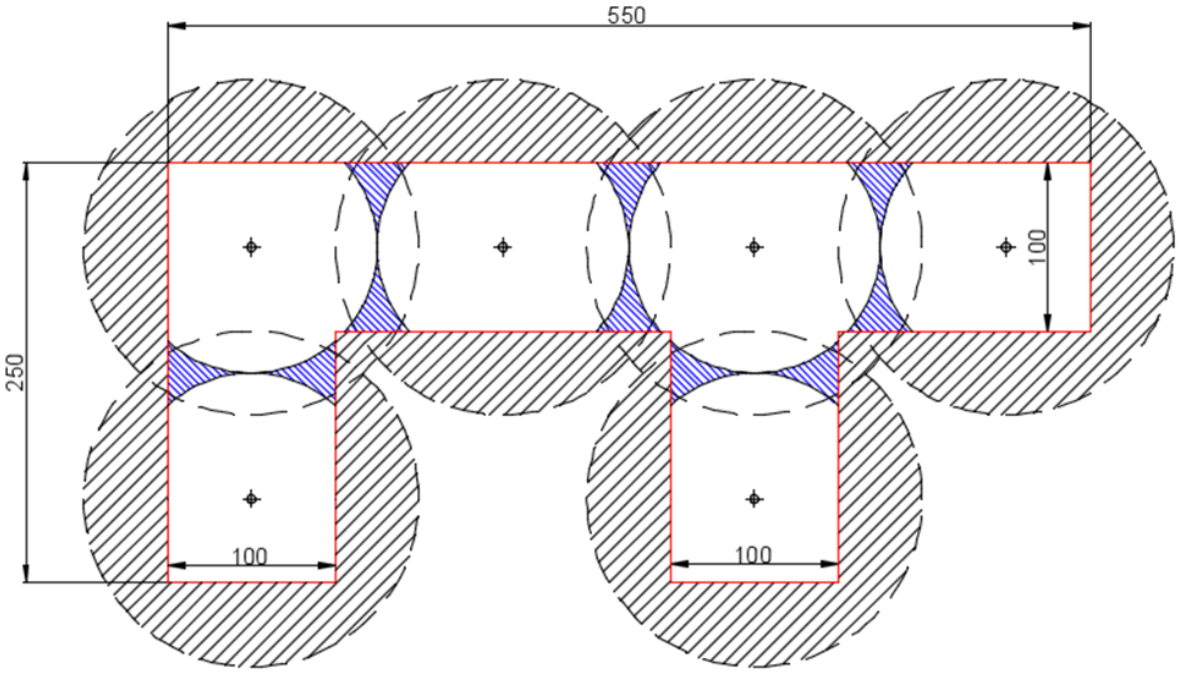

Figura 16 – Disposição dos pontos de acesso e raio de propagação com potência de sinal de $-72dBm$ nos cenários, ilustrando a área aonde aconteceria a desassociação

Com base no cenário descrito na Figura 16, em que não existe área do mapa sem cobertura de algum ponto de acesso e que os pontos de acesso adjacentes possuem uma interseção em seus raios de cobertura, foi constatado que a potência de sinal, no centro desta área de interseção, é de $-72dBm$. Com base nisso, este valor foi utilizado pela controladora para disparar uma mensagem de desassociação.

Como os nós móveis se deslocam aleatoriamente, o mecanismo pode gerar desassociações acima do nível desejado, como, por exemplo, quando o nó se desloca para aquela posição limítrofe, mas não se afasta mais daquele AP e, sim, retorna para mais próximo ao

mesmo. Nesta situação, o nó deveria ter continuado conectado ao mesmo ponto de acesso e, portanto, esse mecanismo adotado seria bem desfavorável, por ainda estar no raio de cobertura do ponto de acesso anteriormente conectado. O mecanismo só é vantajoso ao ocorrer desassociação quando o nó tem a tendência de continuar o envio de dados para um ponto de acesso anteriormente conectado, mas o nó já não está mais no raio de cobertura. Ao transmitir sem sucesso é que se constata tal fato se inicia um processo de busca de um novo ponto de acesso.

4.3 Configurações e parâmetros

Foi utilizado o VMWare Vsphere que virtualiza a CPU e emula o *hardware* do IBM-PC, desde a BIOS até os dispositivos tradicionais, como ambiente hospedeiro das máquinas virtuais. A principal vantagem deste ambiente é que não emula a CPU, o código do sistema-hóspede roda diretamente sobre a CPU real, com velocidade nativa. Portanto, só são virtualizados os demais recursos computacionais.

Para melhorar a velocidade do ambiente foi também instalado as ferramentas da VMWare (VMWare Tools) dentro do sistema hospedeiro, que instala *drivers* otimizados de diversos componentes como vídeo, *clock*, dentre outros. Estes dispositivos normalmente consomem muita CPU para serem emulados, e com os *drivers* especiais ocorre um aumento significativo de desempenho. Estas ferramentas transformam um ambiente virtual num ambiente paravirtual.

A máquina física, hospedeira das máquinas virtuais, foi um Dell PowerEdge R710, que possuía dois processadores de quatro núcleos de 2.8Ghz e 192 GB de RAM.

Foram utilizados duas máquinas virtuais distintas, sendo uma para a controladora RYU e outra para o ambiente integrado MININET/NS-3. O sistema operacional, escolhido para cada máquina virtual, foi o Ubuntu 64-bit Server 14.04.1 LTS, que era a última versão estável à época da implementação. Esse sistema operacional, antes do início dos processos específicos do cenário, possuía um consumo de 1.2GB de memória principal; após o início dos processos, o consumo se mantinha sempre abaixo de 1.6GB.

Inicialmente, cada máquina virtual foi configurada com os seguintes recursos dedicados: um processador de quatro núcleos e 64GB de RAM. Mais tarde, ao analisar o consumo dos recursos e para aumentar o paralelismo das emulações, foram reduzidos os recursos para 4GB de RAM e dois núcleos por máquina virtual e não foram observadas nenhuma alteração de velocidade ou confiabilidade. Tal ação permitiu a execução paralela de ambientes de emulação, a fim de permitir obter os resultados apresentados.

Foram utilizadas nas duas análises das Seções que seguem as seguintes métricas: tempo da desassociação dos nós móveis; e taxa de mensagens Experimenter em relação

às demais mensagens Openflow. Em todos os dois estudos de caso do cenário proposto foram utilizados os parâmetros especificados na Tabela 4.

Tabela 4 – Parâmetros utilizados nos estudos de casos

Tipo do parâmetro	**Valor do parâmetro**
Tempo de emulação	600 s[1]
Interface de rede sem fio	802.11g
Taxa de transferência	54 Mbps
Raio de transmissão	100 metros
Rodadas de emulação para cada ponto	30
Nível de confiança :	95%
Modelos de mobilidade	Random Walk 2D
Velocidade dos nós móveis	0 a 1,0 m/s
Tamanho do mapa	85.000 m^2
Tráfego gerado	100 MB
Número de AP	4 a 6
Número de Nós	2 a 3

O cenário com baixa mobilidade é caracterizado com um baixo envio de mensagens FFRESDWN pois apenas um nó se movimentando e o outro nó é fixo, ou seja, possuindo poucas mudança de características e de conectividade com os pontos de acesso.

O cenário com alota mobilidade é caracterizado com um alto envio de mensagens FFRESDWN pois todos os nós se movimentando, ou seja, possuindo muitas mudança de características e de conectividade com os pontos de acesso o que gera um alto volume de informações sendo enviados para a controladora.

4.4 Análise do Cenário com baixa mobilidade

Nesta primeira análise do cenário, além dos 6 APś, foram acrescentados dois nós, conforme visto na Figura 17, na qual o nó H1 é fixo e o nó H2 se movimenta aleatoriamente por uma região retangular, podendo estar no raio de cobertura de pelo menos um AP ou até de dois $AP's$, podendo ser dos seguintes $AP's$: S1, S2, S3 e S4. Há troca de tráfego UDP entre os dois nós, gerando um tráfego CBR (Constant Bit Rate) a partir do utilitário IPERF[2] ao se transferir um arquivo de 100MB entre os *hosts*. O nó H1 executa um IPERF servidor UDP e o nó H2 executa um IPERF cliente UDP, que se conecta ao servidor do H1 e transfere o arquivo.

Durante a emulação, o nó H2 que, estava posicionado inicialmente no alcance do AP S2, se movimenta para fora do raio de cobertura do AP S2 e, portanto, sofre de-

[2] IPERF é um software livre, do tipo client/server desenvolvido pelo National Laboratory for Applied Network Research (NLANR). Com ele podemos testar/medir o throughput da rede, pois é um gerador de tráfego. É uma ferramenta multiplataforma e está sendo disponibilizada em <https://iperf.fr/iperf-download.php>.

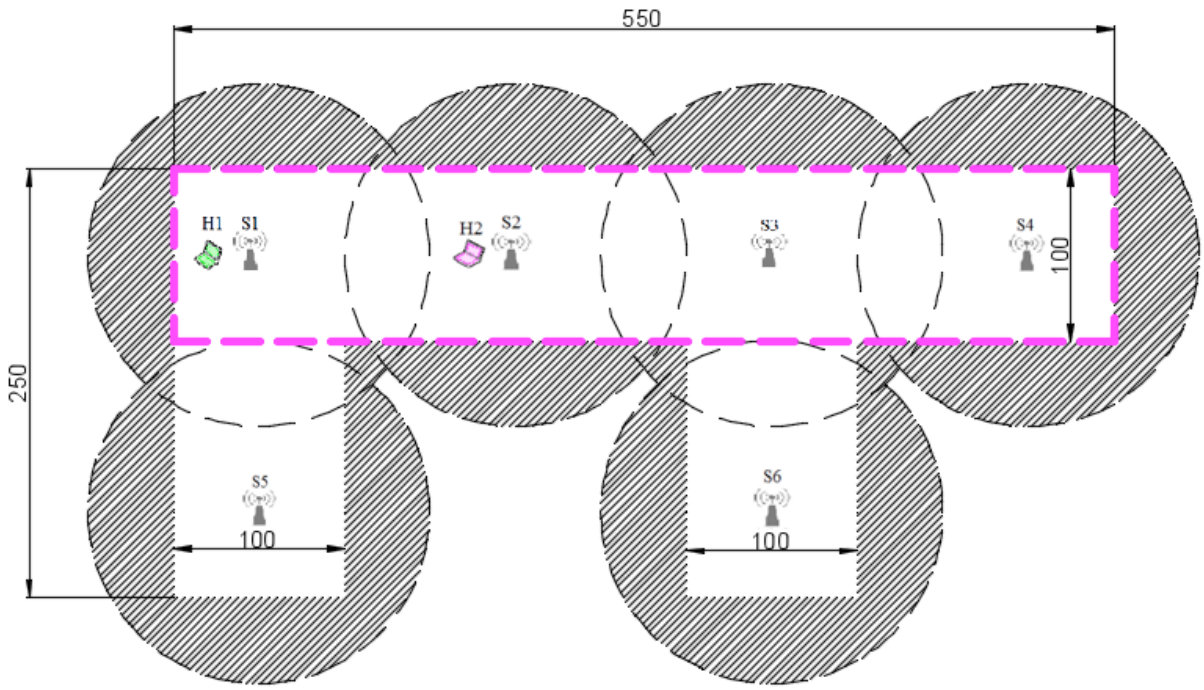

Figura 17 – Disposição do primeiro cenário com baixa mobilidade, com troca de dados entre dois nós no ambiente emulado, sendo H1 um nó estático e H2 um nó com mobilidade

sassociação. Assim H2 iniciará uma varredura para detectar qual outro ponto de acesso irá se associar. Como detalhado anteriormente, o mecanismo de prova de conceito implementado antecipa esse momento de desassociação do nó, a fim de avaliar o impacto do envio das mensagens Experimenter sem confirmação e, também, ver o comportamento do mecanismo de acordo com a média dos experimentos.

Neste cenário com baixa mobilidade, o envio das mensagens Experimenter sem confirmação ocorre quando há mudança das características da rede sem fio do nó H2 e isto é informado pelos pontos de acesso para a controladora. A controladora toma a decisão de mandar uma mensagem de desassociação para o ponto de acesso ao atingir uma queda no nível do sinal para valores menores que $-72dBm$. Foi definida a duração de cada desassociação como sendo o intervalo de tempo decorrido desde o último envio de um *frame* com sucesso, entre o nó e o ponto de acesso, até o instante do próximo envio com sucesso. Em suma, considera-se o tempo do envio com falha (fora do alcance do AP), acrescido do tempo de busca das potências dos $AP's$ na proximidade e o tempo de associação a este novo AP.

Para ilustrar essa decisão da controladora de solicitar uma desassociação antecipada quando ocorre a mobilidade e se atinge o valor acima citado, foi traçado um gráfico de apenas uma das rodados deste cenário com baixa mobilidade, conforme podemos observar na Figura 18. Nesta situação, observamos que o uso do FRESDWN permitiu uma recuperação 29% mais rápida.

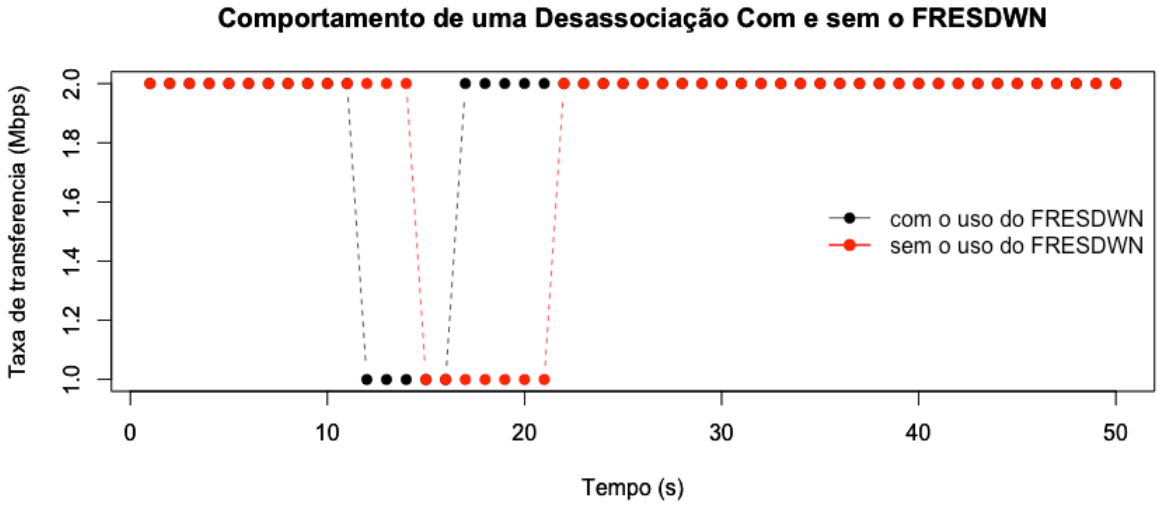

Figura 18 – Disposição do primeiro cenário com baixa mobilidade, com troca de dados entre dois nós no ambiente emulado, sendo H1 um nó estático e H2 um nó com mobilidade

Nas rodadas de simulação do cenário com baixa mobilidade foram observados de uma a cinco desassociações. Para realizar o tratamento estatístico homogêneo entre os números de desassociações, foram realizados várias rodadas de simulação e considerados apenas as trinta primeiras rodadas para cada número de desassociação observado.

Conforme visto na Figura 19, podemos observar que a média das durações das desassociações tem um comportamento crescente e não linear. Até três desassociações não se podia afirmar se o mecanismo era vantajoso, mas esse ganho pode ser observado a partir de 4 desassociações. Era esperado que o comportamento e o tempo de uma desassociação, em relação a múltiplas desassociações, se mantivesse uma diferença que aumenta linearmente, mas foi observado um distanciamento entre o uso do FRESDWN e o SDN tradicional. No SDN tradicional, observam-se valores próximos de um crescimento escalar dentro do nível de confiança. Com o FRESDWN observou-se pequenos ganhos no desempenho, o que o distanciou do outro mecanismo com o aumento das desassociações. Esse ganho deve ter ocorrido de acordo com a mobilidade do nó aliado com as características de perda na propagação do meio físico. Tais fatores, no ambiente emulado, favoreceram esse mecanismo implementado na controladora ao utilizar um critério mais complexo, levando em conta das informações disponibilizadas pelo FRESDWN, no estudo de caso foi utilizado a potência de sinal.

Conforme a Figura 20, podemos observar que o percentual das mensagens Experimenter, em relação às demais mensagens Openflow recebidas e enviadas pela controladora, cresce de maneira linear. Isto era esperado, pois à medida que é necessário informar as mudanças de estados da rede sem fio para a controladora, é enviado uma mensagem Experimenter. Nesta figura é apenas separado a mensagem Experimenter de todas as demais

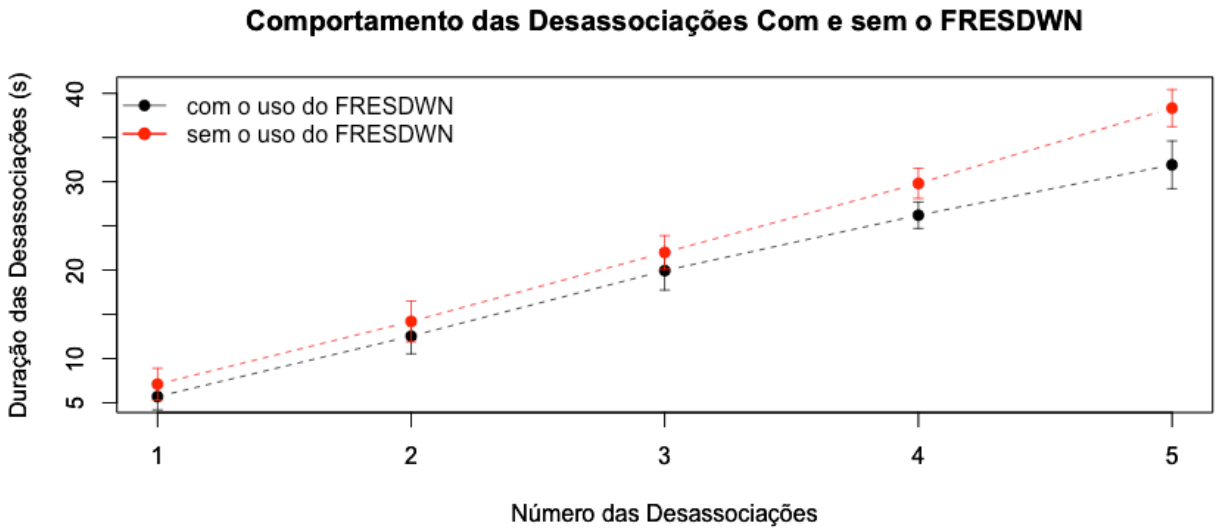

Figura 19 – Resultado dos tempos totais das desassociações do nó H2 no cenário com baixa mobilidade

mensagens suportadas pelo Openflow e analizado o *overhead* que o uso do FRESDWN está gerando no sistema. A Tabela 5 complementa tal análise apresentando os valores absolutos do número de mensagens de controle analisadas.

Tabela 5 – Valores absolutos das mensagens Openflow em relação ao Número de desassociações

desassociações	**Mensagem Experimenter**			**Demais Mensagens Openflow**		
	Mínimo	**Média**	**Máximo**	**Mínimo**	**Média**	**Máximo**
1	3	4	5	106	108	109
2	9	10	11	211	214	217
3	34	36	37	316	322	325
4	72	73	75	423	431	440
5	127	128	130	506	510	511

Ao analisar os resultados da Figura 19 e da Figura 20, podemos observar que a medida que se aumenta o número de desassociações é minimizado o tempo de desassociação. Esperava-se observar que a duração de uma única desassociação fosse proporcional ao ocorrerem 5 desassociação, mas foi observado um aumento de 5 porcento. A utilização do FRESDWN permitiu observar para a partir de quatro desassociações uma redução em relação a não utilização do FRESDWN. Com cinco desassociações observou-se uma redução de 9 % da duração das desassociações quando utilizado o FRESDWN. Em compensação, também aumenta-se proporcionalmente o percentual de mensagens Experimenter sendo recebidas, tratadas e posteriormente enviadas pela controladora. Neste cenário com baixa mobilidade observa-se que possui uma tendência de crescimento do fluxo de mensagens de controle usando o FRESDWN e, por conseguinte, de processamento da controladora.

Isso se observa ao analisarmos que a média de mensagens experimenter para uma única desassociação foi de 4 mensagens FRESDWN e que ao ocorrerem cinco desassociações foi observado 128 mensagens o que representa um aumento de 540%, o que passa de 3,5% das mensagens Openflow totais para uma desassociação para 20% das mensagens Openflow totais ao ser observado cinco desassociações.

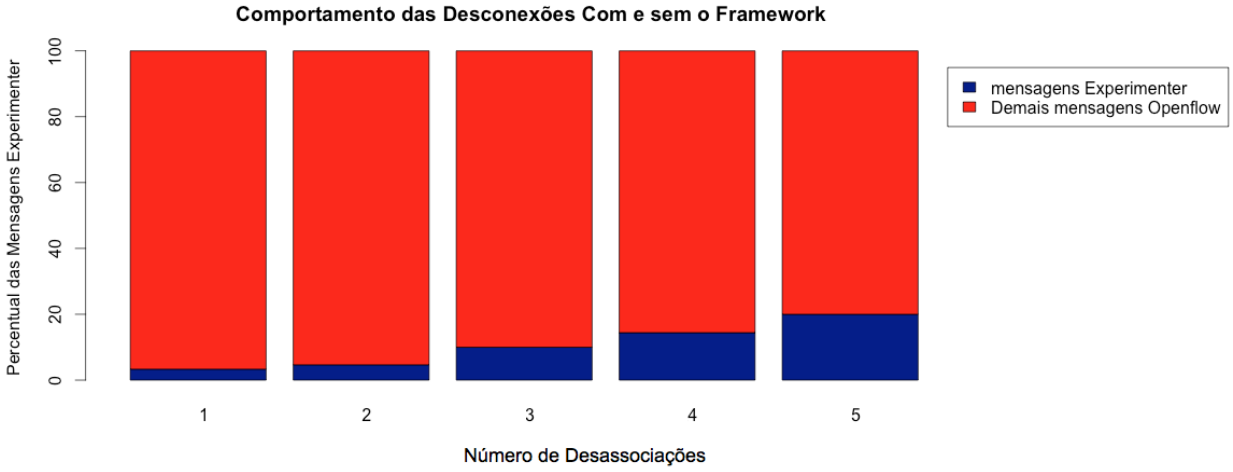

Figura 20 – Percentual de mensagens Experimenter em relação às demais outras mensagens Openflow trocadas pela controladora com os pontos de acesso no cenário com baixa mobilidade

4.4.1 Análise do Cenário com alta mobilidade de nós

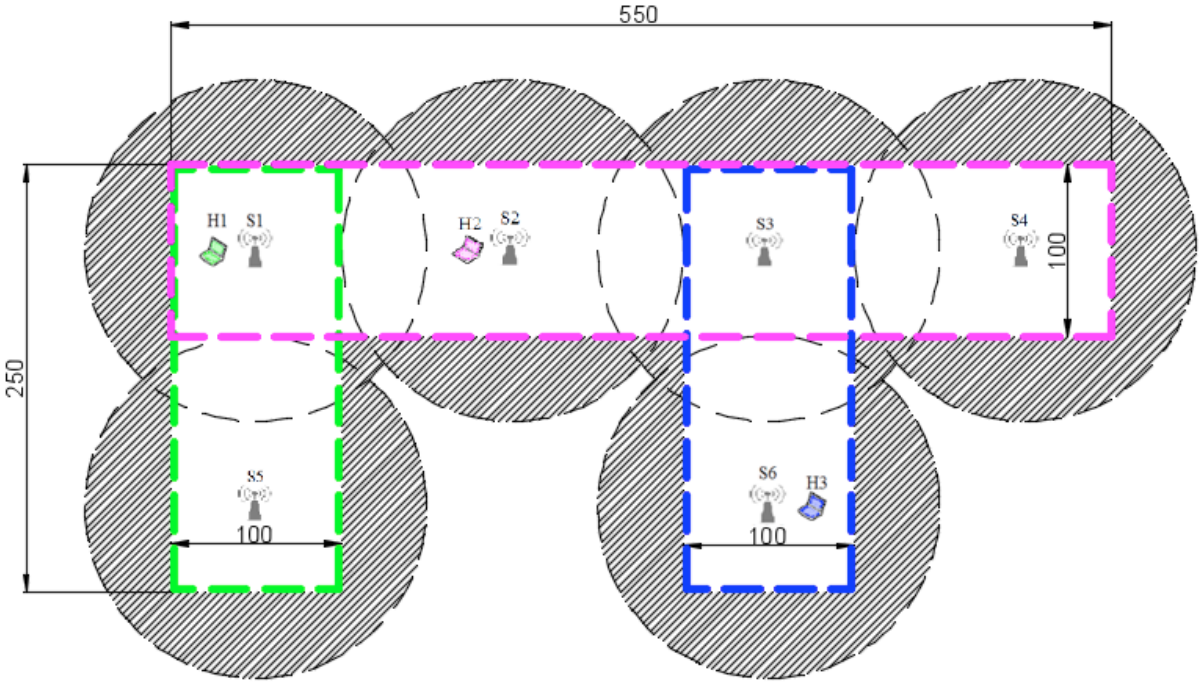

Figura 21 – Disposição do cenário com alta mobilidade com troca de dados entre os três nós móveis no ambiente emulado.

Nesta análise do cenário com alta mobilidade, além dos 6 $AP's$ foram acrescentados três nós móveis, onde todos eles se movimentam aleatoriamente, sendo seu deslocamento delimitado por regiões retangulares conforme a Figura 21. O nó H1 se movimenta sob os raios de cobertura dos $AP's$ S1 e S5; o nó H2 se movimenta sob o raio de cobertura dos $AP's$ S1, S2, S3 e S4; e o nó H3 se movimenta sob os raios de cobertura dos $AP's$ S3 e S6. Há troca de tráfego UDP entre os nós, gerando um tráfego CBR, a partir do utilitário IPERF.

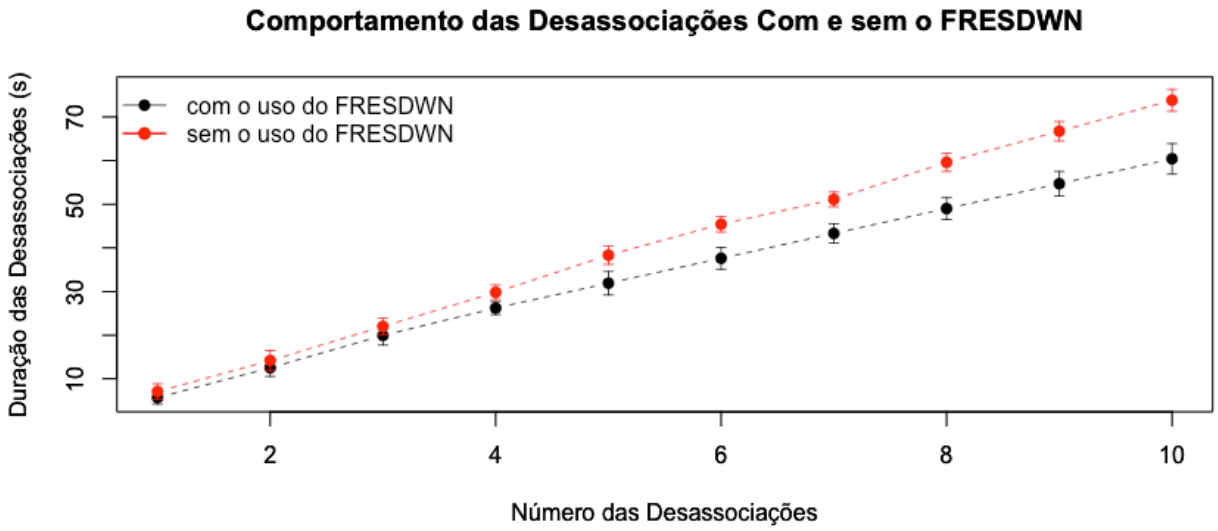

Figura 22 – Resultado dos tempos totais das desassociações do nó H2 no cenário com alta mobilidade

Neste cenário com alta mobilidade, o envio das mensagens Experimenter sem confirmação ocorre quando há mudança das características da rede sem fio de qualquer um dos nós móveis, e isto é informado pelos pontos de acesso para a controladora. De forma análoga à anterior, a controladora toma a decisão de mandar uma mensagem de desassociação para o ponto de acesso ao atingir uma queda no nível do sinal para valores menores que $-72dBm$. Foi definida a duração das desassociações como todo o intervalo de tempo decorrido desde o último envio de um *frame* com sucesso, entre o nó e o ponto de acesso, até o instante do próximo envio com sucesso. Em suma, considera-se o tempo do envio com falha (fora do alcance do AP), acrescido do tempo de busca das potências dos $APt's$ na proximidade e o tempo de associação a este novo AP.

Nas rodadas de simulação do cenário com alta mobilidade foram observados de uma a dez desassociações. Para realizar o tratamento estatístico homogêneo entre os números de desassociações, foram realizados várias rodadas de simulação e considerados apenas as trinta primeiras rodadas para cada número de desassociação observado.

Conforme a Figura 22, podemos observar que a média da grandeza tem um comportamento crescente e não linear. Até quatro desassociações não se podia afirmar se o

mecanismo é vantajoso, mas esse ganho pode ser observado, claramente, a partir de 5 desassociações. Era esperado que o comportamento e o tempo de uma desassociação, em relação a múltiplas desassociações, se mantivesse proporcional, mas também, nesta análise com maior mobilidade, foi observada um distanciamento entre o uso do FRESDWN e o SDN tradicional. No SDN tradicional observam-se valores próximos de um crescimento escalar dentro do nível de confiança; já com o FRESDWN observou-se pequenos ganhos no desempenho, o que o distanciou do outro mecanismo com o aumento das desassociações. Esse ganho deve ter ocorrido de acordo com a mobilidade do nó, aliado com as características de perda na propagação do meio físico. Tais fatores, no ambiente emulado favoreceram esse mecanismo implementado na controladora, ao utilizar um critério mais complexo, fazendo uso das informações disponibilizadas pelo FRESDWN.

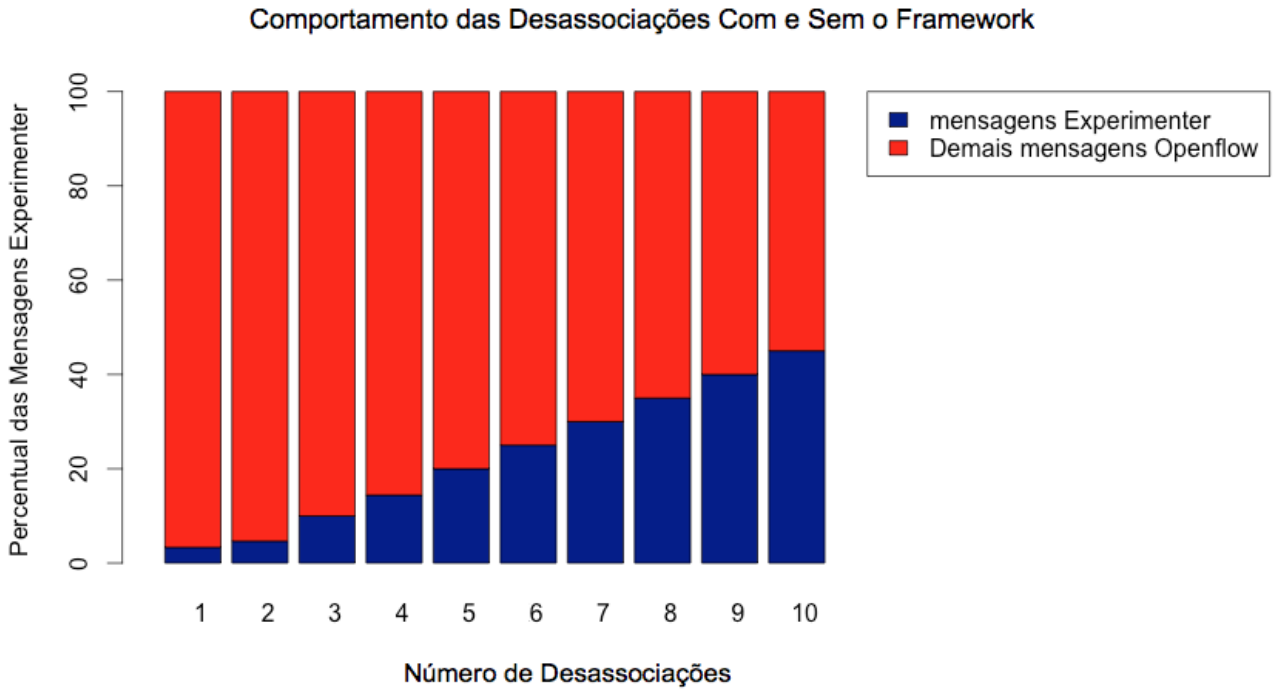

Figura 23 – Taxa de utilização da mensagem Experimenter em relação as demais outras mensagens Openflow trocadas pela controladora com os pontos de acesso no cenário com alta mobilidade

Conforme a Figura 23, podemos observar que o percentual das mensagens Experimenter em relação as demais mensagens Openflow, recebidas e enviadas pela controladora, cresce de maneira linear. Isto era esperado pois, à medida que é necessário informar as mudanças de estados da rede sem fio para a controladora, é enviado uma mensagem Experimenter. Nesta Figura 23 é apenas separado a mensagem Experimenter de todas as demais mensagens suportadas pelo Openflow e analizado o *overhead* que o uso do FRESDWN está gerando no sistema. A Tabela 6 complementa tal análise apresentando os valores absolutos do número de mensagens de controle analisadas, e pode-se observar

que ocorreu apenas uma pequena variação do número mínimo e máximo das mensagens Experimenter. Podemos observar que, com 10 desassociações, ocorreu um crescimento superior a cem vezes mais envios de mensagens experimenter sem confirmação do que ao ocorrer apenas uma desassociação.

Tabela 6 – Valores absolutos das mensagens Openflow em relação ao Número de desassociações

desassociações	**Mensagem Experimenter**			**Demais Mensagens Openflow**		
	Mínimo	**Média**	**Máximo**	**Mínimo**	**Média**	**Máximo**
1	3	4	5	106	108	110
2	9	10	11	209	214	219
3	36	37	38	315	321	328
4	71	72	73	424	432	440
5	126	130	134	501	512	523
6	162	167	171	496	505	515
7	230	237	243	540	550	560
8	303	310	318	572	585	598
9	393	405	417	587	600	614
10	484	501	519	604	620	636

Ao analisar os resultados da Figura 22 e da Figura 23, podemos observar que, a medida que se aumenta o número de desassociações, é minimizado o tempo de desassociação. Esperava-se observar que a duração de uma única desassociação fosse proporcional ao ocorrerem 5 desassociação, mas foi observado um aumento de 7%. A utilização do FRESDWN permitiu observar para a partir de quatro desassociações uma redução em relação a não utilização do FRESDWN. Com dez desassociações observou-se uma redução de 14% da duração das desassociações quando utilizado o FRESDWN. Em compensação, também aumenta-se, proporcionalmente, o percentual de mensagens Experimenter sendo recebidas, tratadas e, posteriormente, enviadas pela controladora. Observa-se que o FRESDWN possui uma tendência de crescimento do fluxo de mensagens de controle e, por conseguinte, de processamento da controladora à medida que se aumenta a mobilidade. Esse aumento não caracteriza um problema e sim um cuidado que se deve tomar em ambientes complexos aonde pode-se utilizar uma controladora distribuída para não ser uma restrição esse aumento do processamento. Isso se observa ao analisarmos que a média de mensagens experimenter para uma única desassociação foi de 4 mensagens FRESDWN e que ao ocorrerem dez desassociações foi observado 501 mensagens o que representa um aumento de 1152% das mensagens FRESDWN, o que passa de 3,5% das mensagens Openflow totais para uma desassociação para 45% das mensagens Openflow totais ao ser observado dez desassociações.

Os resultados obtidos nos permite avaliar que o FRESDWN é viável para estender o Openflow e permitir a inovação em redes sem fio. Também nos permite observar que, para

uma gestão de uma topologia de rede sem fio densa e com muita mobilidade, é necessário estudos complementares que possam analisar outros mecanismos que permitam escalar a capacidade de tratamento da controladora além do citado anteriormente ou que permitam minimizar o número de mensagens Experimenter enviadas para a controladora.

5 Conclusão

Este capítulo apresenta as conclusões do trabalho realizado, destacando as suas contribuições, e propõe alguns trabalhos futuros que podem ser realizados.

5.1 Discussões Finais

As Redes Sem Fio Definidas por Software têm como desafio um ambiente sujeito, constantemente, a mudanças. Estas mudanças podem ocorrer em diversos aspectos das camadas Física e Enlace. Essa característica deve ser considerada pelo plano de controle para lidar com essa particularidade das redes sem fio.

A comunicação móvel é o principal meio de acesso a rede, que utiliza a rede sem fio para sua conexão e continua aumentando expressivamente. Vale relembrar que, no final de 2015, já havia ultrapassado os 7 bilhões de dispositivos móveis (ITU, 2015). Dos dispositivos móveis, destacam-se os *smartphones*, pois se tornaram o padrão para o uso da rede e as suas vendas ultrapassaram 1,2 bilhões de unidades, de um total de 1,8 bilhões de celulares vendidos somente no ano de 2014(GARTNER, 2015).

Os dispositivos sem fio estão cada vez mais utilizando múltiplas tecnologias de rádio transmissão, o que caracteriza as Redes Heterogêneas (TANG; LIAO, 2014). As Redes Sem Fio Heterogêneas possuem os seguintes desafios em SDN: otimização de recursos em ambientes dinâmicos, a granularidade dos controles e das políticas de gerenciamento (OPENNETWORKING.ORG, 2015c).

No nosso entender, o grande desafio reside no aspecto da escalabilidade. Esta pode se referir tanto ao número de elementos da rede sem fio, tamanho da área em termos geográficos, ou quanto à complexidade do cenário em termos do gerenciamento e da administração.

5.2 Contribuições

A contribuição deste trabalho é a proposição do FRESDWN, *framework* que facilita a realização de experimentos em redes sem fio. A especificação do *framework*, permite o desenvolvimento de *softwares* de controle personalizados através da extensão do protocolo Openflow. O mesmo foi estendido através da troca de mensagens Experimenter, a fim de suportar os princípios de rede sem fio.

Esse tipo de mensagem é definido no Openflow como simétrica, e portanto pode ser gerada tanto pela controladora quanto pelo *switch*. Em outras palavras, essas mensagens

são enviadas sem solicitação prévia. O *framework* é flexível o suficiente para que essas mensagens possam ser enviadas, sem ou com confirmação.

O FRESDWN proposto permite a inovação em um ambiente replicável realístico utilizando SDN e permitindo a comunicação entre nós e controladora, o que permite investigar e validar propostas para alguns dos problemas relativos a redes sem fio.

Na implementação da mensagem do FRESDWN, foi otimizada a capacidade de transmissão, utilizando o tamanho típico de MTU que é de 1500 bytes. Permite, após os descontos dos demais cabeçalhos, enviar um vetor de dados Experimenter de até 1.408 bytes, sendo 1.392 úteis para qualquer aplicação que utilize o FRESDWN.

Em relação à implementação do FRESDWN, esta foi dividida em três componentes básicos: a controladora externa Openflow (RYU), o *switch* Openflow customizado (ofsoftswitch13) e o ambiente integrado MININET versão 2.2.1 com NS-3 versão 3.22. Foi realizada uma emulação baseada em simulação ao utilizar o MININET para se comunicar com o controladora RYU e o simulador de redes NS-3 para representar a mobilidade dos nós sem fio e também os componentes da camada física e de enlace.

Além da implementação do FRESDWN no ambiente integrado do MININET mais NS-3, foi necessário a alteração no *switch* Openflow, conforme ANEXO A, e na controladora RYU, conforme ANEXO B, para dar suporte ao envio e recebimento de mensagens Experimenter.

Também foram construídas máquinas virtuais distintas para a controladora RYU, adaptada para tratar mensagens do *framewok* e o ambiente integrado MININET/NS-3 com ofsoftswitch13 customizado para tratar mensagens Experimenter do FRESDWN.

Através dos estudos de caso com baixa e com alta mobilidade, implementamos um mecanismo na controladora de desassociação antecipada, com base nas informações recebidas dos pontos de acesso. Através deste mecanismo, a controladora utilizou as mensagens Experimenter e conseguimos obter resultados que mostraram a viabilidade do FRESDWN.

Foram obtidos resultados do número de desassociações dos nós móveis, em relação ao tempo, e também outros resultados da proporção das mensagens Experimenter em relação às demais mensagens do plano de controle.

5.3 Trabalhos Futuros

Seria conveniente estender o ambiente emulado para suportar, além dos nós emulados, nós reais, ou seja, integrar um ambiente real ao ambiente emulado (um ambiente híbrido). Comparar o comportamento do FRESDWN e seu desempenho nos três cenários: no ambiente emulado, no testbed e no ambiente híbrido. Poderia se estender o FRESDWN para contemplar todas as características relevantes para a controladora do ambiente sem

fio, atendendo plenamente a um cenário heterogêneo, ou seja, suportando todas as multiplicidades de tecnologias sem fio.

Observamos um crescimento significativo da proporção das mensagens Experimenter em relação às demais mensagens do plano de controle, o que gera a necessidade de trabalhos complementares para avaliar o FRESDWN em uma rede densa e com alta mobilidade. Nesta situação, seria necessário analisar mecanismos para aumentar a capacidade da controladora ou minimizar o número de mensagens Experimenter enviadas.

Uma possibilidade seria implementar utilizando controladoras distribuídas, que permitem uma maior escalabilidade, do que as controladoras centralizadas.

Uma outra possibilidade seria o envio de mensagens apenas pela controladora, a fim de solicitar informações sobre o estado dos nós sem fio. Essa abordagem pode ser investigada, mas deve se considerar questões de inconsistência das informações disponíveis na controladora, em relação à atual topologia de rede sem fio.

Podem ser analisados diversos algoritmos na controladora, levando em conta da informação recebida através do FRESDWN. Para se avaliar o desempenho, pode-se comparar esses algoritmos com as abordagens tradicionais.

Sugerimos a implementação do FRESDWN no OVS, pois deve possuir um desempenho superior pois é um switch openflow de Kernel de sistema. Normalmente os processos de Kernel de sistema tendem a ser mais eficiente do que um à nivel de usuários. Deve-se comparar esses desempenhos para auxiliar a tomada de decisão.

Referências

BOULHOSA, Reinaldo; TÁSSIOCARVALHO, José Jailton Junior; PASSOS, Diego; DIAS, Kelvin Lopes; CERQUEIRA, Eduardo. Gerenciamento de handover transparente com suporte integrado a qos/qoe em redes heterogêneas. In: *Congresso da Sociedade Brasileira de Computação. In: X Workshop em Desempenho de Sistemas Computacionais e de Comunicação, Natal*. [S.l.: s.n.], 2011. v. 10, p. 108–122. Citado na página 2.

CAMPBELL, Andrew T.; MEER, Herman G. De; KOUNAVIS, Michael E.; MIKI, Kazuho; VICENTE, John B.; VILLELA, Daniel. A survey of programmable networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 29, n. 2, p. 7–23, abr. 1999. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/505733.505735>. Citado na página 9.

CAMPBELL-KELLY, Martin; GARCIA-SWARTZ, Daniel; LAM, Richard; YANG, Yilei. Economic and business perspectives on smartphones as multi-sided platforms. *Telecommunications Policy*, v. 39, n. 8, p. 717 – 734, 2015. ISSN 0308-5961. Special Issue on Consumer behavior and telecommunications policy. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0308596114001591>. Citado na página 1.

CASADO, Martin; FREEDMAN, Michael J.; PETTIT, Justin; LUO, Jianying; MCKEOWN, Nick; SHENKER, Scott. Ethane: Taking control of the enterprise. In: *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM, 2007. (SIGCOMM '07), p. 1–12. ISBN 978-1-59593-713-1. Disponível em: <http://doi.acm.org/10.1145/1282380.1282382>. Citado 2 vezes nas páginas 18 e 19.

CASADO, Martin; GARFINKEL, Tal; AKELLA, Aditya; FREEDMAN, Michael J.; BONEH, Dan; MCKEOWN, Nick; SHENKER, Scott. Sane: A protection architecture for enterprise networks. In: *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*. Berkeley, CA, USA: USENIX Association, 2006. (USENIX-SS'06). Disponível em: <http://dl.acm.org/citation.cfm?id=1267336.1267346>. Citado 2 vezes nas páginas 18 e 19.

CHAN, Min-Cheng; CHEN, Chien; HUANG, Jun-Xian; KUO, T.; YEN, Li-Hsing; TSENG, Chien-Chao. Opennet: A simulator for software-defined wireless local area network. In: *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*. [S.l.: s.n.], 2014. p. 3332–3336. Citado 2 vezes nas páginas 28 e 36.

CHATER, H.; CHAFNAJI, H. Green telecommunication with heterogonous networks: A survey. In: *Multimedia Computing and Systems (ICMCS), 2014 International Conference on*. [S.l.: s.n.], 2014. p. 1490–1494. Citado na página 1.

CISCO. *Cisco Virtual Wireless Controller*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <http://www.cisco.com/c/en/us/products/wireless/virtual-wireless-controller/index.html>. Citado na página 5.

CISCO. *Wireless LAN Controller*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <http://www.cisco.com/c/en/us/products/wireless/wireless-lan-controller/index.html>. Citado na página 5.

COMMITTEE, ONF Market Education et al. Software-defined networking: The new norm for networks. *ONF White Paper*, 2012. Citado 2 vezes nas páginas 1 e 3.

CPQD. *Openflow 1.3 Software Switch*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://github.com/CPqD/ofsoftswitch13>. Citado na página 6.

CROCKFORD, D. *The application/json Media Type for JavaScript Object Notation (JSON)*. IETF, 2006. RFC 4627 (Informational). (Request for Comments, 4627). Obsoleted by RFC 7159. Disponível em: <http://www.ietf.org/rfc/rfc4627.txt>. Citado na página 39.

D-LINK. *Software Managed Wireless Access Points*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <http://us.dlink.com/business-solutions/wireless/software-managed-wireless-access-points/>. Citado na página 5.

D-LINK. *Unified Wireless: Controllers and Switches*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <http://us.dlink.com/business-solutions/wireless/unified-wireless-controllers-and-switches/>. Citado na página 5.

DETTI, A.; PISA, C.; SALSANO, S.; BLEFARI-MELAZZI, N. Wireless mesh software defined networks (wmsdn). In: *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. [S.l.: s.n.], 2013. p. 89–95. ISSN 2160-4886. Citado na página 26.

DORIA, A.; SALIM, J. Hadi; HAAS, R.; KHOSRAVI, H.; WANG, W.; DONG, L.; GOPAL, R.; HALPERN, J. *Forwarding and Control Element Separation (ForCES) Protocol Specification*. IETF, 2010. RFC 5810 (Proposed Standard). (Request for Comments, 5810). Updated by RFCs 7121, 7391. Disponível em: <http://www.ietf.org/rfc/rfc5810.txt>. Citado 2 vezes nas páginas 11 e 17.

FONTES, Ramon R; AFZAL, Samira; BRITO, Samuel HB; SANTOS, Mateus AS; ROTHENBERG, Christian Esteve. Mininet-wifi: Emulating software-defined wireless networks. In: IEEE. *Network and Service Management (CNSM), 2015 11th International Conference on*. [S.l.], 2015. p. 384–389. Citado na página 27.

GARTNER. *Gartner Says Smartphone Sales Surpassed One Billion Units in 2014*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <http://www.gartner.com/newsroom/id/2996817>. Citado 3 vezes nas páginas 1, 2 e 59.

HALEPLIDIS, E.; DENAZIS, S.; KOUFOPAVLOU, O.; HALPERN, J.; SALIM, J.H. Software-defined networking: Experimenting with the control to forwarding plane interface. In: *Software Defined Networking (EWSDN), 2012 European Workshop on*. [S.l.: s.n.], 2012. p. 91–96. Citado na página 18.

HALEPLIDIS, E.; PENTIKOUSIS, K.; DENAZIS, S.; SALIM, J. Hadi; MEYER, D.; KOUFOPAVLOU, O. *Software-Defined Networking (SDN): Layers and Architecture Terminology*. IETF, 2015. RFC 7426 (Informational). (Request for Comments, 7426). [Online; acessado em 15-janeiro-2016]. Disponível em: <http://www.ietf.org/rfc/rfc7426.txt>. Citado 3 vezes nas páginas x, 10 e 18.

HALEPLIDIS, E.; SALIM, J.H.; HALPERN, J.M.; HARES, S.; PENTIKOUSIS, K.; OGAWA, K.; WEIMING, Wang; DENAZIS, S.; KOUFOPAVLOU, O. Network programmability with forces. *Communications Surveys Tutorials, IEEE*, v. 17, n. 3, p. 1423–1440, thirdquarter 2015. ISSN 1553-877X. Citado 4 vezes nas páginas x, 16, 17 e 18.

HALPERN, J.; SALIM, J. Hadi. *Forwarding and Control Element Separation (ForCES) Forwarding Element Model*. IETF, 2010. RFC 5812 (Proposed Standard). (Request for Comments, 5812). Updated by RFC 7408. Disponível em: <http://www.ietf.org/rfc/rfc5812.txt>. Citado na página 17.

Han, Q.; Ferreira, P.; Costeira, J. P. Asymmetric Peer Influence in Smartphone Adoption in a Large Mobile Network. *ArXiv e-prints*, jan. 2016. Citado na página 1.

HAN, S.; ZHANG, X.; SHIN, K. Fair and efficient coexistence of heterogeneous channel widths in next-generation wireless lans. *Mobile Computing, IEEE Transactions on*, PP, n. 99, p. 1–1, 2016. ISSN 1536-1233. Citado na página 5.

HAQUE, I. T.; ABU-GHAZALEH, N. Wireless software defined networking: a survey and taxonomy. *IEEE Communications Surveys Tutorials*, PP, n. 99, p. 1–1, 2016. ISSN 1553-877X. Citado na página 30.

HP. *Wireless LAN*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.hpe.com/us/en/networking/wlan.html>. Citado na página 5.

IEEE. *IEEE Standard for Information technology– Telecommunications and information exchange between systems Local and metropolitan area networks*. 2013. [Online; acessado em 15-janeiro-2016]. Disponível em: <http://standards.ieee.org/getieee802/download/802.11ac-2013.pdf>. Citado na página 5.

ITU, International Telecommunication Union. *The World in 2015: ICT Facts and Figures*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <http://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx>. Citado 2 vezes nas páginas 1 e 59.

JANG, Yu-Teng Jacky; CHANG, Shuchih Ernest; SHEN, Wei-Cheng; WANG, Sheng-Wen. Cloud backup: an enhanced smartphone app designed with cross-platform approach. *International Journal of Embedded Systems*, Inderscience Publishers (IEL), v. 8, n. 2-3, p. 174–184, 2016. Citado na página 40.

LEI, Lei; ZHONG, Zhangdui; ZHENG, Kan; CHEN, Jiadi; MENG, Hanlin. Challenges on wireless heterogeneous networks for mobile cloud computing. *Wireless Communications, IEEE*, v. 20, n. 3, p. 34–44, June 2013. ISSN 1536-1284. Citado na página 4.

LIANG, Chengchao; YU, F.R. Wireless network virtualization: A survey, some research issues and challenges. *Communications Surveys Tutorials, IEEE*, v. 17, n. 1, p. 358–380, Firstquarter 2015. ISSN 1553-877X. Citado na página 2.

MCKEOWN, Nick; ANDERSON, Tom; BALAKRISHNAN, Hari; PARULKAR, Guru; PETERSON, Larry; REXFORD, Jennifer; SHENKER, Scott; TURNER, Jonathan. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833.

Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>. Citado 4 vezes nas páginas 3, 11, 19 e 21.

MENDONCA, Marc; ASTUTO, Bruno Nunes; OBRACZKA, Katia; TURLETTI, Thierry. Software Defined Networking for Heterogeneous Networks. *IEEE MMTC E-Letters*, IEEE, v. 8, n. 3, p. 36–39, maio 2013. Disponível em: <https://hal.inria.fr/hal-00838709>. Citado na página 4.

MONTEIRO, Alex; SOUTO, Eduardo; PAZZI, Richard; KILJANDER, Jussi. Atribuição dinâmica de canais em redes sem fio não coordenadas ieee 802.11, baseada em fatores de sobreposição e intensidade de sinal. Citado na página 2.

NAUDTS, Bram; KIND, Mario; VERBRUGGE, Sofie; COLLE, Didier; PICKAVET, Mario. How can a mobile service provider reduce costs with software-defined networking? *International Journal of Network Management*, v. 26, n. 1, p. 56–72, 2016. ISSN 1099-1190. Disponível em: <http://dx.doi.org/10.1002/nem.1919>. Citado na página 4.

NIEPHAUS, C.; GHINEA, G.; ALIU, O.G.; HADZIC, S.; KRETSCHMER, M. Sdn in the wireless context - towards full programmability of wireless network elements. In: *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. [S.l.: s.n.], 2015. p. 1–6. Citado na página 3.

NSF.GOV. *Campus Cyberinfrastructure - Data, Networking, and Innovation Program | NSF - National Science Foundation*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=504748>. Citado na página 3.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.0.0 (31/12/2009)*. 2009. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-spec-v1.0.0.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.1.0 (28/02/2011)*. 2011. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-spec-v1.1.0.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.2 (01/12/2011)*. 2011. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-spec-v1.2.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.3.0 (25/06/2012)*. 2012. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-spec-v1.3.0.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.3.1 (6/09/2012)*. 2012. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-spec-v1.3.1.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.3.2 (25/04/2013)*. 2013. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-spec-v1.0.0.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.3.3(18/12/2013)*. 2013. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-spec-v1.3.3.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.4.0 (15/10/2013)*. 2013. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-spec-v1.4.0.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.3.4 (27/03/2014)*. 2014. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-switch-v1.3.4.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.5.0 (19/12/2014)*. 2014. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-switch-v1.5.0.noipr.pdf>. Citado 2 vezes nas páginas 6 e 24.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.3.5 (26/03/2015)*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-switch-v1.3.5.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Openflow Switch Specification 1.5.1 (26/03/2015)*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/Openflow/Openflow-switch-v1.5.1.pdf>. Citado na página 6.

OPENNETWORKING.ORG. *Solution Brief: Openflow-Enabled Mobile and Wireless Networks - Open Networking Foundation*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/solution-brief-Openflow-enabled-mobile-and-wireless-networks>. Citado 2 vezes nas páginas 4 e 59.

OPENNETWORKING.ORG. *The Evolution of SDN and Openflow: A Standards Perspective*. 2016. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/IEEE-papers/evolution-of-sdn-and-of.pdf>. Citado na página 4.

OPENWISP.ORG. *OpenWisp - Wireless Everywhere*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <http://openwisp.org/>. Citado na página 5.

OPENWRT.ORG. *OpenWrt - Wireless Freedom*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://openwrt.org/>. Citado na página 5.

PÖTSCH, Albert. A scalable testbed infrastructure for embedded industrial wireless sensor and actuator networks: Ph. d. forum abstract. In: IEEE PRESS. *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. [S.l.], 2016. p. 45. Citado na página 40.

RNP. *Serviço Eduroam (education roaming)*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://portal.rnp.br/web/servicos/eduroam>. Citado na página 5.

SÁNCHEZ, Carlos Fernández. Openflow 1.3 with ryu controller. Universitat Politècnica de Catalunya, 2015. Citado na página 6.

SCHOENWAELDER, J. *Overview of the 2002 IAB Network Management Workshop*. IETF, 2003. RFC 3535 (Informational). (Request for Comments, 3535). Disponível em: <http://www.ietf.org/rfc/rfc3535.txt>. Citado na página 12.

SDXCENTRAL. *Infographic: SDN market size to reach $35B by 2018*. 2015. [Online; acessado em 15-janeiro-2016]. Disponível em: <https://www.sdxcentral.com/reports/sdn-market-size-infographic-2013/>. Citado na página 3.

SEZER, S.; SCOTT-HAYWARD, S.; CHOUHAN, P.K.; FRASER, B.; LAKE, D.; FINNEGAN, J.; VILJOEN, N.; MILLER, M.; RAO, N. Are we ready for sdn? implementation challenges for software-defined networks. *Communications Magazine, IEEE*, v. 51, n. 7, p. 36–43, July 2013. ISSN 0163-6804. Citado na página 1.

SURESH, Lalith; SCHULZ-ZANDER, Julius; MERZ, Ruben; FELDMANN, Anja; VAZAO, Teresa. Towards programmable enterprise wlans with odin. In: ACM. *Proceedings of the first workshop on Hot topics in software defined networks*. [S.l.], 2012. p. 115–120. Citado 3 vezes nas páginas x, 25 e 43.

TANG, Wenqiang; LIAO, Qing. An sdn-based approach for load balance in heterogeneous radio access networks. In: *Computer Applications and Communications (SCAC), 2014 IEEE Symposium on*. [S.l.: s.n.], 2014. p. 105–108. Citado 2 vezes nas páginas 4 e 59.

WU, Shuai; HAO, Jack Jianxiu; XIA, Wei; JIN, Zhiying. *Orchestration server for video distribution network*. [S.l.]: Google Patents, 2016. US Patent 9,307,042. Citado na página 40.

XIA, Wenfeng; WEN, Yonggang; FOH, Chuan Heng; NIYATO, D.; XIE, Haiyong. A survey on software-defined networking. *Communications Surveys Tutorials, IEEE*, v. 17, n. 1, p. 27–51, Firstquarter 2015. ISSN 1553-877X. Citado 2 vezes nas páginas xii e 9.

XIA, Wenfeng; WEN, Yonggang; FOH, Chuan Heng; NIYATO, D.; XIE, Haiyong. A survey on software-defined networking. *Communications Surveys Tutorials, IEEE*, v. 17, n. 1, p. 27–51, Firstquarter 2015. ISSN 1553-877X. Citado na página 4.

YAN, Muxi; CASEY, Jasson; SHOME, Prithviraj; SPRINTSON, Alex; SUTTON, Andrew. {\ AE} therflow: Principled wireless support in sdn. *arXiv preprint arXiv:1509.04745*, 2015. Citado na página 29.

YANG, L.; DANTU, R.; ANDERSON, T.; GOPAL, R. *Forwarding and Control Element Separation (ForCES) Framework*. IETF, 2004. RFC 3746 (Informational). (Request for Comments, 3746). Disponível em: <http://www.ietf.org/rfc/rfc3746.txt>. Citado 2 vezes nas páginas 16 e 17.

YINGXIONG, Li; CHAN, Henry CB; LAM, Patrick; CHONG, Peter HJ. Channel allocation method for multi-radio wireless mesh networks based on a genetic algorithm. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. [S.l.: s.n.], 2016. v. 2. Citado na página 40.

YUVARADNI, Bhujbal; DHANAHSRI, Darandale; SONALI, Gunjal; GAURI, Tekale; THITE, Mr Sandip. Health monitoring services using wireless body area network. *Imperial Journal of Interdisciplinary Research*, v. 2, n. 5, 2016. Citado na página 40.

Anexos

ANEXO A – ofsoftswitch13

O Openflow 1.3 Software Switch é uma implementação no espaço de endereçamento de usuário de switch Openflow. O seu código é baseado na implementação do Ericsson TrafficLab 1.1, com o suporte no plano de encaminhamento para a versão 1.3 do Openflow.

É desmembrado nos seguintes principais componentes:

1. ofdatapath: a implementação do switch propriamente dito
2. ofprotocol: canal seguro de comunicação entre o switch e a controladora
3. oflib: a biblioteca que converte de/para o formato 1.3
4. oflib-exp: a biblioteca que converte as mensagens experimenter de/para o formato 1.3
5. dpctl: ferramentas para configurar o switch pela console

Foram necessários fazer ajustes nos códigos a seguir para prover o suporte básico a troca de mensagens.

A.1 Makefile.in

```
160 oflib-exp/ofl-exp-fresdwn.o oflib-exp/ofl-exp-nicira.o \

...

221 am_oflib_exp_liboflib_exp_a_OBJECTS = oflib-exp/ofl-exp.$(OBJEXT) \
222 oflib-exp/ofl-exp-fresdwn.$(OBJEXT) \
223 oflib-exp/ofl-exp-nicira.$(OBJEXT) \
```

Código fonte 1 – Alterações do Makefile.in do Ofsoftswitch13

A.2 Makefile

```
152 lib_libopenflow_a_DEPENDENCIES = oflib/ofl-actions.o \
153 ^^Ioflib/ofl-actions-pack.o oflib/ofl-actions-print.o \
154 ^^Ioflib/ofl-actions-unpack.o oflib/ofl-messages.o \
155 ^^Ioflib/ofl-messages-pack.o oflib/ofl-messages-print.o \
156 ^^Ioflib/ofl-messages-unpack.o oflib/ofl-structs.o \
157 ^^Ioflib/ofl-structs-match.o oflib/ofl-structs-pack.o \
```

```
158 ^^Ioflib/ofl-structs-print.o oflib/ofl-structs-unpack.o \
159 ^^Ioflib/oxm-match.o oflib/ofl-print.o oflib-exp/ofl-exp.o \
160 ^^Ioflib-exp/ofl-exp-fresdwn.o oflib-exp/ofl-exp-nicira.o \
161 ^^Ioflib-exp/ofl-exp-openflow.o

    ...

221 am_oflib_exp_liboflib_exp_a_OBJECTS = oflib-exp/ofl-exp.$(OBJEXT) \
222 ^^Ioflib-exp/ofl-exp-fresdwn.$(OBJEXT) \
223 ^^Ioflib-exp/ofl-exp-nicira.$(OBJEXT) \
224 ^^Ioflib-exp/ofl-exp-openflow.$(OBJEXT)

    ...

244 am__udatapath_libudatapath_a_SOURCES_DIST = udatapath/action_set.c \
245 ^^Iudatapath/action_set.h udatapath/crc32.c udatapath/crc32.h \
246 ^^Iudatapath/datapath.c udatapath/datapath.h \
247 ^^Iudatapath/dp_actions.c udatapath/dp_actions.h \
248 ^^Iudatapath/dp_buffers.c udatapath/dp_buffers.h \
249 ^^Iudatapath/dp_control.c udatapath/dp_control.h \
250 ^^Iudatapath/dp_exp.c udatapath/dp_exp.h udatapath/dp_fresdwn.c \
251 ^^Iudatapath/dp_fresdwn.h udatapath/flow_table.c \
252 ^^Iudatapath/flow_table.h udatapath/flow_entry.c \
253 ^^Iudatapath/flow_entry.h udatapath/group_table.c \
254 ^^Iudatapath/group_table.h udatapath/group_entry.c \
255 ^^Iudatapath/group_entry.h udatapath/match_std.c \
256 ^^Iudatapath/match_std.h udatapath/packet.c udatapath/packet.h \
257 ^^Iudatapath/packet_handle_std.c udatapath/packet_handle_std.h \
258 ^^Iudatapath/pipeline.c udatapath/pipeline.h \
259 ^^Iudatapath/udatapath.c

    ...

293 am_udatapath_ofdatapath_OBJECTS =  \
294 ^^Iudatapath/udatapath_ofdatapath-action_set.$(OBJEXT) \
295 ^^Iudatapath/udatapath_ofdatapath-crc32.$(OBJEXT) \
296 ^^Iudatapath/udatapath_ofdatapath-datapath.$(OBJEXT) \
297 ^^Iudatapath/udatapath_ofdatapath-dp_actions.$(OBJEXT) \
298 ^^Iudatapath/udatapath_ofdatapath-dp_buffers.$(OBJEXT) \
299 ^^Iudatapath/udatapath_ofdatapath-dp_control.$(OBJEXT) \
300 ^^Iudatapath/udatapath_ofdatapath-dp_exp.$(OBJEXT) \
301 ^^Iudatapath/udatapath_ofdatapath-dp_fresdwn.$(OBJEXT) \
302 ^^Iudatapath/udatapath_ofdatapath-dp_ports.$(OBJEXT) \
303 ^^Iudatapath/udatapath_ofdatapath-flow_table.$(OBJEXT) \
304 ^^Iudatapath/udatapath_ofdatapath-flow_entry.$(OBJEXT) \
305 ^^Iudatapath/udatapath_ofdatapath-group_table.$(OBJEXT) \
306 ^^Iudatapath/udatapath_ofdatapath-group_entry.$(OBJEXT) \
307 ^^Iudatapath/udatapath_ofdatapath-match_std.$(OBJEXT) \
308 ^^Iudatapath/udatapath_ofdatapath-meter_entry.$(OBJEXT) \
```

```
309 ^^Iudatapath/udatapath_ofdatapath-meter_table.$(OBJEXT) \
310 ^^Iudatapath/udatapath_ofdatapath-packet.$(OBJEXT) \
311 ^^Iudatapath/udatapath_ofdatapath-packet_handle_std.$(OBJEXT) \
312 ^^Iudatapath/udatapath_ofdatapath-pipeline.$(OBJEXT) \
313 ^^Iudatapath/udatapath_ofdatapath-udatapath.$(OBJEXT)

    ...

693 ACLOCAL = ${SHELL} /home/gerpe/FRESDWN/ofsoftswitch13/build-aux/missing aclocal-1.14
694 AMTAR = $${TAR-tar}
695 AM_DEFAULT_VERBOSITY = 1
696 AUTOCONF = ${SHELL} /home/gerpe/FRESDWN/ofsoftswitch13/build-aux/missing autoconf
697 AUTOHEADER = ${SHELL} /home/gerpe/FRESDWN/ofsoftswitch13/build-aux/missing autoheader
698 AUTOMAKE = ${SHELL} /home/gerpe/FRESDWN/ofsoftswitch13/build-aux/missing automake-1.14

    ...

733 MAKEINFO = ${SHELL} /home/gerpe/FRESDWN/ofsoftswitch13/build-aux/missing makeinfo

    ...

757 abs_builddir = /home/gerpe/FRESDWN/ofsoftswitch13
758 abs_srcdir = /home/gerpe/FRESDWN/ofsoftswitch13
759 abs_top_builddir = /home/gerpe/FRESDWN/ofsoftswitch13
760 abs_top_srcdir = /home/gerpe/FRESDWN/ofsoftswitch13

    ...

780 install_sh = ${SHELL} /home/gerpe/FRESDWN/ofsoftswitch13/build-aux/install-sh

    ...

873 noinst_HEADERS = include/openflow/nicira-ext.h \
874 ^^Iinclude/openflow/fresdwn-ext.h include/openflow/private-ext.h \
875 ^^Iinclude/openflow/openflow.h \
876 ^^Iinclude/openflow/openflow-netlink.h

    ...

908 lib_libopenflow_a_LIBADD = oflib/ofl-actions.o \
909                            oflib/ofl-actions-pack.o \
910                            oflib/ofl-actions-print.o \
911                            oflib/ofl-actions-unpack.o \
912                            oflib/ofl-messages.o \
913                            oflib/ofl-messages-pack.o \
914                            oflib/ofl-messages-print.o \
915                            oflib/ofl-messages-unpack.o \
916                            oflib/ofl-structs.o \
917 ^^I^^I^^I               oflib/ofl-structs-match.o \
918                            oflib/ofl-structs-pack.o \
919                            oflib/ofl-structs-print.o \
```

```
920                             oflib/ofl-structs-unpack.o \
921                             oflib/oxm-match.o \
922                             oflib/ofl-print.o \
923                             oflib-exp/ofl-exp.o \
924                             oflib-exp/ofl-exp-fresdwn.o \
925                             oflib-exp/ofl-exp-nicira.o \
926                             oflib-exp/ofl-exp-openflow.o
```

...

```
963 oflib_exp_liboflib_exp_a_SOURCES = \
964 ^^Ioflib-exp/ofl-exp.c \
965 ^^Ioflib-exp/ofl-exp.h \
966     oflib-exp/ofl-exp-fresdwn.c \
967     oflib-exp/ofl-exp-fresdwn.h \
968 ^^Ioflib-exp/ofl-exp-nicira.c \
969 ^^Ioflib-exp/ofl-exp-nicira.h \
970 ^^Ioflib-exp/ofl-exp-openflow.c \
971 ^^Ioflib-exp/ofl-exp-openflow.h
```

...

```
1006 udatapath_ofdatapath_SOURCES = \
1007 ^^Iudatapath/action_set.c \
1008 ^^Iudatapath/action_set.h \
1009 ^^Iudatapath/crc32.c \
1010 ^^Iudatapath/crc32.h \
1011 ^^Iudatapath/datapath.c \
1012 ^^Iudatapath/datapath.h \
1013 ^^Iudatapath/dp_actions.c \
1014 ^^Iudatapath/dp_actions.h \
1015 ^^Iudatapath/dp_buffers.c \
1016 ^^Iudatapath/dp_buffers.h \
1017 ^^Iudatapath/dp_control.c \
1018 ^^Iudatapath/dp_control.h \
1019 ^^Iudatapath/dp_exp.c \
1020 ^^Iudatapath/dp_exp.h \
1021 ^^Iudatapath/dp_fresdwn.c \
1022 ^^Iudatapath/dp_fresdwn.h \
1023 ^^Iudatapath/dp_ports.c \
1024 ^^Iudatapath/dp_ports.h \
1025 ^^Iudatapath/flow_table.c \
1026 ^^Iudatapath/flow_table.h \
1027 ^^Iudatapath/flow_entry.c \
1028 ^^Iudatapath/flow_entry.h \
1029 ^^Iudatapath/group_table.c \
1030 ^^Iudatapath/group_table.h \
1031 ^^Iudatapath/group_entry.c \
1032 ^^Iudatapath/group_entry.h \
```

```
1033 ^^Iudatapath/match_std.c \
1034     udatapath/match_std.h \
1035 ^^Iudatapath/meter_entry.c \
1036 ^^Iudatapath/meter_entry.h \
1037 ^^Iudatapath/meter_table.c \
1038 ^^Iudatapath/meter_table.h \
1039 ^^Iudatapath/packet.c \
1040 ^^Iudatapath/packet.h \
1041 ^^Iudatapath/packet_handle_std.c \
1042     udatapath/packet_handle_std.h \
1043 ^^Iudatapath/pipeline.c \
1044 ^^Iudatapath/pipeline.h \
1045 ^^Iudatapath/udatapath.c

...

1250 oflib-exp/ofl-exp-fresdwn.$(OBJEXT): oflib-exp/$(am__dirstamp) \
1251 ^^Ioflib-exp/$(DEPDIR)/$(am__dirstamp)

...

1322 udatapath/udatapath_libudatapath_a-dp_fresdwn.$(OBJEXT):  \
1323 ^^Iudatapath/$(am__dirstamp) udatapath/$(DEPDIR)/$(am__dirstamp

...

1432 udatapath/udatapath_ofdatapath-dp_fresdwn.$(OBJEXT):  \
1433 ^^Iudatapath/$(am__dirstamp) udatapath/$(DEPDIR)/$(am__dirstamp)

...

1699 include oflib-exp/$(DEPDIR)/ofl-exp-fresdwn.Po

...

1733 include udatapath/$(DEPDIR)/udatapath_libudatapath_a-dp_fresdwn.Po

...

1750 include udatapath/$(DEPDIR)/udatapath_ofdatapath-dp_fresdwn.Po

...

1883 udatapath/udatapath_libudatapath_a-dp_fresdwn.o: udatapath/dp_fresdwn.c
1884 ^^I$(AM_V_CC)$(CC) $(DEFS) $(DEFAULT_INCLUDES) $(INCLUDES)
     ↪  $(udatapath_libudatapath_a_CPPFLAGS) $(CPPFLAGS) $(AM_CFLAGS) $(CFLAGS) -MT
     ↪  udatapath/udatapath_libudatapath_a-dp_fresdwn.o -MD -MP -MF
     ↪  udatapath/$(DEPDIR)/udatapath_libudatapath_a-dp_fresdwn.Tpo -c -o
     ↪  udatapath/udatapath_libudatapath_a-dp_fresdwn.o `test -f 'udatapath/dp_fresdwn.c'
     ↪  || echo '$(srcdir)/'`udatapath/dp_fresdwn.c
1885 ^^I$(AM_V_at)$(am__mv) udatapath/$(DEPDIR)/udatapath_libudatapath_a-dp_fresdwn.Tpo
     ↪  udatapath/$(DEPDIR)/udatapath_libudatapath_a-dp_fresdwn.Po
```

```
1886 #^^I$(AM_V_CC)source='udatapath/dp_fresdwn.c'
     ↪  object='udatapath/udatapath_libudatapath_a-dp_fresdwn.o' libtool=no \
1887 #^^IDEPDIR=$(DEPDIR) $(CCDEPMODE) $(depcomp) \
1888 #^^I$(AM_V_CC_no)$(CC) $(DEFS) $(DEFAULT_INCLUDES) $(INCLUDES)
     ↪  $(udatapath_libudatapath_a_CPPFLAGS) $(CPPFLAGS) $(AM_CFLAGS) $(CFLAGS) -c -o
     ↪  udatapath/udatapath_libudatapath_a-dp_fresdwn.o `test -f 'udatapath/dp_fresdwn.c'
     ↪  || echo '$(srcdir)/'`udatapath/dp_fresdwn.c
1889
1890 udatapath/udatapath_libudatapath_a-dp_fresdwn.obj: udatapath/dp_fresdwn.c
1891 ^^I$(AM_V_CC)$(CC) $(DEFS) $(DEFAULT_INCLUDES) $(INCLUDES)
     ↪  $(udatapath_libudatapath_a_CPPFLAGS) $(CPPFLAGS) $(AM_CFLAGS) $(CFLAGS) -MT
     ↪  udatapath/udatapath_libudatapath_a-dp_fresdwn.obj -MD -MP -MF
     ↪  udatapath/$(DEPDIR)/udatapath_libudatapath_a-dp_fresdwn.Tpo -c -o
     ↪  udatapath/udatapath_libudatapath_a-dp_fresdwn.obj `if test -f
     ↪  'udatapath/dp_fresdwn.c'; then $(CYGPATH_W) 'udatapath/dp_fresdwn.c'; else
     ↪  $(CYGPATH_W) '$(srcdir)/udatapath/dp_fresdwn.c'; fi`
1892 ^^I$(AM_V_at)$(am__mv) udatapath/$(DEPDIR)/udatapath_libudatapath_a-dp_fresdwn.Tpo
     ↪  udatapath/$(DEPDIR)/udatapath_libudatapath_a-dp_fresdwn.Po
1893 #^^I$(AM_V_CC)source='udatapath/dp_fresdwn.c'
     ↪  object='udatapath/udatapath_libudatapath_a-dp_fresdwn.obj' libtool=no \
1894 #^^IDEPDIR=$(DEPDIR) $(CCDEPMODE) $(depcomp) \
1895 #^^I$(AM_V_CC_no)$(CC) $(DEFS) $(DEFAULT_INCLUDES) $(INCLUDES)
     ↪  $(udatapath_libudatapath_a_CPPFLAGS) $(CPPFLAGS) $(AM_CFLAGS) $(CFLAGS) -c -o
     ↪  udatapath/udatapath_libudatapath_a-dp_fresdwn.obj `if test -f
     ↪  'udatapath/dp_fresdwn.c'; then $(CYGPATH_W) 'udatapath/dp_fresdwn.c'; else
     ↪  $(CYGPATH_W) '$(srcdir)/udatapath/dp_fresdwn.c'; fi`
     ...

2121 udatapath/udatapath_ofdatapath-dp_fresdwn.o: udatapath/dp_fresdwn.c
2122 ^^I$(AM_V_CC)$(CC) $(DEFS) $(DEFAULT_INCLUDES) $(INCLUDES)
     ↪  $(udatapath_ofdatapath_CPPFLAGS) $(CPPFLAGS) $(AM_CFLAGS) $(CFLAGS) -MT
     ↪  udatapath/udatapath_ofdatapath-dp_fresdwn.o -MD -MP -MF
     ↪  udatapath/$(DEPDIR)/udatapath_ofdatapath-dp_fresdwn.Tpo -c -o
     ↪  udatapath/udatapath_ofdatapath-dp_fresdwn.o `test -f 'udatapath/dp_fresdwn.c' ||
     ↪  echo '$(srcdir)/'`udatapath/dp_fresdwn.c
2123 ^^I$(AM_V_at)$(am__mv) udatapath/$(DEPDIR)/udatapath_ofdatapath-dp_fresdwn.Tpo
     ↪  udatapath/$(DEPDIR)/udatapath_ofdatapath-dp_fresdwn.Po
2124 #^^I$(AM_V_CC)source='udatapath/dp_fresdwn.c'
     ↪  object='udatapath/udatapath_ofdatapath-dp_fresdwn.o' libtool=no \
2125 #^^IDEPDIR=$(DEPDIR) $(CCDEPMODE) $(depcomp) \
2126 #^^I$(AM_V_CC_no)$(CC) $(DEFS) $(DEFAULT_INCLUDES) $(INCLUDES)
     ↪  $(udatapath_ofdatapath_CPPFLAGS) $(CPPFLAGS) $(AM_CFLAGS) $(CFLAGS) -c -o
     ↪  udatapath/udatapath_ofdatapath-dp_fresdwn.o `test -f 'udatapath/dp_fresdwn.c' ||
     ↪  echo '$(srcdir)/'`udatapath/dp_fresdwn.c
2127
2128 udatapath/udatapath_ofdatapath-dp_fresdwn.obj: udatapath/dp_fresdwn.c
```

```
2129 ^^I$(AM_V_CC)$(CC) $(DEFS) $(DEFAULT_INCLUDES) $(INCLUDES)
     ↪  $(udatapath_ofdatapath_CPPFLAGS) $(CPPFLAGS) $(AM_CFLAGS) $(CFLAGS) -MT
     ↪  udatapath/udatapath_ofdatapath-dp_fresdwn.obj -MD -MP -MF
     ↪  udatapath/$(DEPDIR)/udatapath_ofdatapath-dp_fresdwn.Tpo -c -o
     ↪  udatapath/udatapath_ofdatapath-dp_fresdwn.obj `if test -f
     ↪  'udatapath/dp_fresdwn.c'; then $(CYGPATH_W) 'udatapath/dp_fresdwn.c'; else
     ↪  $(CYGPATH_W) '$(srcdir)/udatapath/dp_fresdwn.c'; fi`
2130 ^^I$(AM_V_at)$(am__mv) udatapath/$(DEPDIR)/udatapath_ofdatapath-dp_fresdwn.Tpo
     ↪  udatapath/$(DEPDIR)/udatapath_ofdatapath-dp_fresdwn.Po
2131 #^^I$(AM_V_CC)source='udatapath/dp_fresdwn.c'
     ↪  object='udatapath/udatapath_ofdatapath-dp_fresdwn.obj' libtool=no \
2132 #^^IDEPDIR=$(DEPDIR) $(CCDEPMODE) $(depcomp) \
2133 #^^I$(AM_V_CC_no)$(CC) $(DEFS) $(DEFAULT_INCLUDES) $(INCLUDES)
     ↪  $(udatapath_ofdatapath_CPPFLAGS) $(CPPFLAGS) $(AM_CFLAGS) $(CFLAGS) -c -o
     ↪  udatapath/udatapath_ofdatapath-dp_fresdwn.obj `if test -f
     ↪  'udatapath/dp_fresdwn.c'; then $(CYGPATH_W) 'udatapath/dp_fresdwn.c'; else
     ↪  $(CYGPATH_W) '$(srcdir)/udatapath/dp_fresdwn.c'; fi`
```

Código fonte 2 – Alterações do Makefile do Ofsoftswitch13

A.3 lib

A.3.1 lib/automake.mk

```
109 oflib-exp/ofl-exp-fresdwn.o \
110 oflib-exp/ofl-exp-nicira.o \
```

Código fonte 3 – Alterações do lib/automake.mk

A.4 ofl-exp

A.4.1 ofl-exp/ofl-exp.c

```
38 #include "ofl-exp-fresdwn.h"

   ...

45 #include "openflow/fresdwn-ext.h"
```

Código fonte 4 – Acrescentar em ofl-exp.c

A.4.2 ofl-exp/ofl-exp-fresdwn.h

```
1 #ifndef OFL_EXP_FRESDWN_H
2 #define OFL_EXP_FRESDWN_H 1
```

```
3  /*
4   *
5   *  FRESDWN
6   *
7   */
8
9  #include "../oflib/ofl-structs.h"
10 #include "../oflib/ofl-messages.h"
11
12 struct ofl_exp_fresdwn_msg_header {
13     struct ofl_msg_experimenter   header; /* FRESDWN_VENDOR_ID */
14     uint32_t   type;
15 };
16
17 struct ofl_exp_fresdwn_msg_dummy {
18     struct ofl_exp_fresdwn_msg_header   header; /* FRESDWN_VENDOR_ID */
19     uint8_t dummy_content[1400];
20 };
21
22 struct ofl_exp_fresdwn_msg_role {
23     struct ofl_exp_fresdwn_msg_header   header; /* FRESDWNT_ROLE_REQUEST|REPLY */
24     uint32_t role;
25 };
26
27 int
28 ofl_exp_fresdwn_msg_pack(struct ofl_msg_experimenter *msg, uint8_t **buf, size_t
    ↪  *buf_len);
29
30 ofl_err
31 ofl_exp_fresdwn_msg_unpack(struct ofp_header *oh, size_t *len, struct
    ↪  ofl_msg_experimenter **msg);
32
33 int
34 ofl_exp_fresdwn_msg_free(struct ofl_msg_experimenter *msg);
35
36 char *
37 ofl_exp_fresdwn_msg_to_string(struct ofl_msg_experimenter *msg);
38
39 #endif
```

Código fonte 5 – ofl-exp-fresdwn.h

A.4.3 oflib-exp/.deps/ofl-exp.Po

```
58 oflib-exp/../oflib/ofl-structs.h oflib-exp/ofl-exp-fresdwn.h \

...
```

```
71 include/openflow/fresdwn-ext.h
```

Código fonte 6 – Alterações do oflib-exp/.deps/ofl-exp.Po

A.4.4 ofl-exp/ofl-exp-fresdwn.c

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <netinet/in.h>
4
5  #include "openflow/openflow.h"
6  #include "openflow/fresdwn-ext.h"
7  #include "ofl-exp-fresdwn.h"
8  #include "../oflib/ofl-print.h"
9  #include "../oflib/ofl-log.h"
10
11 #define LOG_MODULE ofl_exp_fdwn
12 OFL_LOG_INIT(LOG_MODULE)
13
14 /*
15  *
16  *  FRESDWN
17  *
18  */
19
20 int
21 ofl_exp_fresdwn_msg_pack(struct ofl_msg_experimenter *msg, uint8_t **buf, size_t
    ↪  *buf_len) {
22     struct fresdwn_header *fw;
23     struct ofl_exp_fresdwn_msg_header *exp = (struct ofl_exp_fresdwn_msg_header *)msg;
24     switch (exp->type) {
25         case (FRESDWNT_DUMMY):{
26             struct ofl_exp_fresdwn_msg_dummy *dummy = (struct
                ↪  ofl_exp_fresdwn_msg_dummy*) exp;
27             struct fresdwn_dummy *ofp_dummy;
28
29             *buf_len = sizeof(struct fresdwn_dummy);
30             *buf     = (uint8_t *)malloc(*buf_len);
31             ofp_dummy = (struct fresdwn_dummy*)(*buf);
32
33             /* Copy the array contents*/
34             memcpy(ofp_dummy->dummy_content, dummy->dummy_content,
                ↪  FRESDWN_ARRAY_SIZE);
35             break;
36         }
37         case (FRESDWNT_FEATURE_REQUEST):{
38             return 0;
```

```
39          }
40          case (FRESDWNT_FEATURE_REPLY):{
41              return 0;
42          } // IDENTIFICADOR 1
43          default: {
44              OFL_LOG_WARN(LOG_MODULE, "Trying to print unknown FRESDWN Experimenter
                ↪  message.");
45              return -1;
46          }
47      }
48      /* Pack experimenter header */
49      fw = (struct fresdwn_header *)(*buf);
50      fw->fdwnh.experimenter = htonl(exp->header.experimenter_id);
51      fw->fdwnh.exp_type =  htonl(exp->type);
52
53      return 0;
54  }
55
56  ofl_err
57  ofl_exp_fresdwn_msg_unpack(struct ofp_header *oh, size_t *len, struct
    ↪  ofl_msg_experimenter **msg) {
58  /*
59      Criadas para tentar enviar a mensagem experimenter
60      uint8_t **buf = NULL;
61      size_t *buf_len = NULL;
62  */
63      struct fresdwn_header *exp;
64  /*
65   * parando de ignorar o tamanho errado
66  */
67      if (*len < sizeof(struct fresdwn_header)) {
68          OFL_LOG_WARN(LOG_MODULE, "Received EXPERIMENTER message has invalid length
            ↪  (%zu).", *len);
69          return ofl_error(OFPET_BAD_REQUEST, OFPBRC_BAD_LEN);
70      }
71  /*
72  * parando de ignorar o tamanho errado
73  */
74      exp = (struct fresdwn_header *)oh;
75      switch (ntohl(exp->fdwnh.exp_type)) {
76          case (FRESDWNT_DUMMY):{
77              struct fresdwn_dummy *src;
78              struct ofl_exp_fresdwn_msg_dummy *dst;
79
80  /*
81  * parando de ignorar o tamanho errado
82  */
```

```
83              if (*len < sizeof(struct fresdwn_dummy)) {
84                  OFL_LOG_WARN(LOG_MODULE, "Received DUMMY message has invalid length
                    ↪  (%zu).", *len);
85                  return ofl_error(OFPET_BAD_REQUEST, OFPBRC_BAD_LEN);
86              }
87
88  /*
89  * parando de ignorar o tamanho errado
90  */
91              *len -= sizeof(struct fresdwn_dummy);
92              src = (struct fresdwn_dummy *)exp;
93              dst = (struct ofl_exp_fresdwn_msg_dummy *)malloc(sizeof(struct
                ↪  ofl_exp_fresdwn_msg_dummy));
94
95              dst->header.header.experimenter_id = ntohl(exp->fdwnh.experimenter);
96              dst->header.type = ntohl(exp->fdwnh.exp_type);
97              memcpy(dst->dummy_content, src->dummy_content, FRESDWN_ARRAY_SIZE);
98
99              (*msg) = (struct ofl_msg_experimenter *)dst;
100
101             return 0;
102         }
103         case (FRESDWNT_FEATURE_REQUEST):{
104             return 0;
105         }
106         case (FRESDWNT_FEATURE_REPLY): {
107             return 0;
108         }
109         // IDENTIFICADOR 1
110         //case (FRESDWNT_STATUS_REQUEST): // IDENTIFICADOR 1
111         case (FRESDWNT_STATUS_REPLY):{
112             return 0;
113         } // IDENTIFICADOR 1
114         default: {
115             OFL_LOG_WARN(LOG_MODULE, "Trying to unpack unknown FRESDWN Experimenter
                ↪  message.");
116             //return ofl_error(OFPET_BAD_REQUEST, OFPBRC_BAD_EXPERIMENTER);
117             return ofl_error(OFPET_BAD_REQUEST, OFPBRC_BAD_EXP_TYPE);
118         }
119     }
120     // (*msg).experimenter_id = exp
121     free(msg);
122     return 0;
123 }
124
125 int
126 ofl_exp_fresdwn_msg_free(struct ofl_msg_experimenter *msg) {
```

```
127     if (msg->experimenter_id == FRESDWN_VENDOR_ID) {
128         struct ofl_exp_fresdwn_msg_header *exp = (struct ofl_exp_fresdwn_msg_header
            ↪  *)msg;
129         switch (exp->type) {
130             case (FRESDWNT_DUMMY):{
131                 break;
132             }
133             default: {
134                 OFL_LOG_WARN(LOG_MODULE, "Trying to free unknown FRESDWN Experimenter
                    ↪  message.");
135             }
136         }
137     } else {
138         OFL_LOG_WARN(LOG_MODULE, "Trying to free non-FRESDWN Experimenter message.");
139     }
140     free(msg);
141     return 0;
142 }
143 
144 char *
145 ofl_exp_fresdwn_msg_to_string(struct ofl_msg_experimenter *msg) {
146     char *str;
147     size_t str_size;
148     FILE *stream = open_memstream(&str, &str_size);
149     if (msg->experimenter_id == FRESDWN_VENDOR_ID) {
150         struct ofl_exp_fresdwn_msg_header *exp = (struct ofl_exp_fresdwn_msg_header
            ↪  *)msg;
151         switch (exp->type) {
152             case FRESDWNT_DUMMY:{
153                 OFL_LOG_WARN(LOG_MODULE, "Printing a dummy message");
154                 fprintf(stream, "ofexp{type=\"%u\"}", exp->type);
155             }
156             default: {
157                 OFL_LOG_WARN(LOG_MODULE, "Trying to print unknown FRESDWN Experimenter
                    ↪  message.");
158                 fprintf(stream, "ofexp{type=\"%u\"}", exp->type);
159             }
160         }
161     } else {
162         OFL_LOG_WARN(LOG_MODULE, "Trying to print non-FRESDWN Experimenter message.");
163         fprintf(stream, "exp{exp_id=\"%u\"}", msg->experimenter_id);
164     }
165 
166     fclose(stream);
167     return str;
168 }
```

Código fonte 7 – ofl-exp/ofl-exp-fresdwn.c

A.4.5 oflib-exp/automake.mk

```
6  oflib-exp/ofl-exp-fresdwn.c \
7  oflib-exp/ofl-exp-fresdwn.h \
```

Código fonte 8 – Alterações do oflib-exp/automake.mk

A.5 udatapath

A.5.1 udatapath/dp$_f resdwn.h$

```
1  #ifndef DP_FRESDWN_H
2  #define DP_FRESDWN_H 1
3
4  #include "datapath.h"
5  #include "oflib-exp/ofl-exp-fresdwn.h"
6
7  ofl_err
8  dp_fresdwn_handle_dummy(struct datapath *dp, struct ofl_exp_fresdwn_msg_dummy
   ↪  *msg,const struct sender *sender);
9
10 #endif
```

Código fonte 9 – dp$_f resdwn.h$

A.5.2 udatapath/dp$_f resdwn.c$

```
6  #include "dp_fresdwn.h"
7  #include "openflow/fresdwn-ext.h"
8  #include <jansson.h>
9
10 ofl_err
11 dp_fresdwn_handle_dummy(struct datapath *dp, struct ofl_exp_fresdwn_msg_dummy
   ↪  *msg,const struct sender *sender){
12  /*
13      *   testava se estava zerado
14      */
15  /*
16     uint8_t *dummy_arr;
17     uint8_t test_block[FRESDWN_ARRAY_SIZE];
18  */
19     struct ofl_exp_fresdwn_msg_dummy reply = {{.type = FRESDWNT_DUMMY},
20              .dummy_content = {0}};
```

```
21
22      /* Nesting craziness... consider a change to the name of the struct field name */
23      reply.header.header.experimenter_id = FRESDWN_VENDOR_ID;
24      reply.header.header.header.type = OFPT_EXPERIMENTER;
25
26
27          memcpy(reply.dummy_content, msg->dummy_content, FRESDWN_ARRAY_SIZE);
28          dp_send_message(dp, (struct ofl_msg_header *)&reply, sender);
29
30        /* Send error. You can define your own error codes, since this one do not make
          ↪  sense and is just an illustrative example*/
31      return ofl_error(OFPET_QUEUE_OP_FAILED, OFPQOFC_BAD_PORT);
32  }
```

Código fonte 10 – udatapath/dp$_f resdwn.c$

A.5.3 udatapath/automake.mk

```
23  udatapath/dp_fresdwn.c \
24  udatapath/dp_fresdwn.h \
```

Código fonte 11 – Alterações do udatapath/automake.mk

A.5.4 udatapath/dp$_e xp.c$

```
36  #include "dp_fresdwn.h"

    ...

44  #include "oflib-exp/ofl-exp-fresdwn.h"
```

Código fonte 12 – Alterações do udatapath/dp$_e xp.c$

A.5.5 udatapath/.deps/udatapath$_o f datapath - dp_e xp.Po$

```
69  /usr/include/x86_64-linux-gnu/bits/posix2_lim.h udatapath/dp_fresdwn.h \
70  oflib-exp/ofl-exp-fresdwn.h include/openflow/openflow-ext.h \
71  include/openflow/fresdwn-ext.h lib/vlog.h lib/vlog-modules.def \
```

Código fonte 13 – Alterações do udatapath/.deps/udatapath$_o f datapath - dp_e xp.Po$

A.6 include/openflow

A.6.1 include/openflow/fresdwn-ext.h

```
1  #include "openflow/openflow.h"
2
3  #define FRESDWN_OUI_STR "0x00000005"
4  #define FRESDWN_VENDOR_ID 0x00000005
5  #define FRESDWN_ARRAY_SIZE 1392
6  /*
7   *
8   *      FRESDWN
9   *
10  */
11
12 struct fresdwn_header {
13     struct ofp_experimenter_header fdwnh;
14 };
15 OFP_ASSERT(sizeof(struct fresdwn_header) == 16);
16
17 /* Freswdn messages types*/
18 enum fresdwn_type {
19     /* Switch status request.  The request body is an ASCII string that
20      * specifies a prefix of the key names to include in the output; if it is
21      * the null string, then all key-value pairs are included. */
22     // FRESDWNT_STATUS_REQUEST,
23     FRESDWNT_FEATURE_REQUEST, // Solicita as caracteristicas do dispositivo
24
25     /* Switch status reply.  The reply body is an ASCII string of key-value
26      * pairs in the form "key=value\n". */
27     FRESDWNT_FEATURE_REPLY, // Informa as caracteristicas para a controladora
28     FRESDWNT_STATUS_REQUEST,
29
30     /* Switch status reply.  The reply body is an ASCII string of key-value
31      * pairs in the form "key=value\n". */
32     FRESDWNT_STATUS_REPLY,
33
34     /* Generic GET/SET Wireless configuration */
35     FRESDWNT_ACT_SET_CONFIG,
36     FRESDWNT_ACT_GET_CONFIG,
37     FRESDWNT_COMMAND_REQUEST,
38     FRESDWNT_COMMAND_REPLY,
39     FRESDWNT_FLOW_END_CONFIG,
40     FRESDWNT_FLOW_END,
41     FRESDWNT_MGMT,
42
43     /* GET/SET Channel configuration */
```

```
44      FRESDWNT_ACT_SET_CHANNEL_CONFIG,
45      FRESDWNT_ACT_GET_CHANNEL_CONFIG,
46
47      /* GET PHYSICAL information */
48      FRESDWNT_ACT_GET_AP_CONFIG,
49      FRESDWNT_ACT_GET_AP_IN_RANGE_INFO,
50      FRESDWNT_ACT_GET_BEACON_INFO,
51      FRESDWNT_ACT_GET_NOISE_INFO,
52      FRESDWNT_ACT_GET_POSITION_INFO,
53
54      /* GET/SET SSID configuration */
55      FRESDWNT_ACT_SET_SSID_CONFIG,
56      FRESDWNT_ACT_GET_SSID_CONFIG,
57
58      /* DUMMY Message for tests*/
59      FRESDWNT_DUMMY
60  };
61
62  struct fresdwn_dummy {
63      struct fresdwn_header header;
64      uint8_t dummy_content[FRESDWN_ARRAY_SIZE];
65  };
66  OFP_ASSERT(sizeof(struct fresdwn_dummy) == 1408);
67
68  /* Wireless technology types*/
69  enum fresdwn_type_wireless_tecnology {
70      FRESDWNT_802_11_a,
71      FRESDWNT_802_11_b,
72      FRESDWNT_802_11_g,
73      FRESDWNT_802_11_n,
74      FRESDWNT_802_11_ac,
75      FRESDWNT_3G,
76      FRESDWNT_4G,
77      FRESDWNT_5G
78  };
79
80  enum fresdwn_type_of_wireless_atribute {
81  // 802.11
82      FRESDWNT_FREQUENCY, // 2,4gH
83      FRESDWNT_CHANNEL_SIZE, // 20mhz
84      FRESDWNT_CHANNEL_NUMBER, // 6
85      FRESDWNT_TRANSMITION_TYPE, // FHSS
86      FRESDWNT_TRANSMITION_MIN_POWER, // 3mW
87      FRESDWNT_TRANSMITION_POWER, // 30mW
88      FRESDWNT_TRANSMITION_MAX_POWER, // 30mW
89      FRESDWNT_SSID, // CONVIDADO - 32chars
90      FRESDWNT_BSSID, // 8f:....
```

```
91      FRESDWNT_BSSID_TYPE, // 1-infrastructure, 2-independent 3-any
92      FRESDWNT_CRYPTO_TECNOLOGY, // WPA
93      FRESDWNT_CRYPTO_TYPE_OF_KEY, // TPIK
94      FRESDWNT_CRYPTO_SIZE_OF_KEY, // 256bits
95      FRESDWNT_ANTENNA_POWER, // 2dbi
96      FRESDWNT_ANTENNA_TYPE, // omnidirecional
97      FRESDWNT_ANTENNA_QUANTITIES, // omnidirecional
98      FRESDWNT_DATA_RATE_RX, // 11M
99      FRESDWNT_DATA_RATE_SUPPORTED_RX, // 1M,11M,54M   126chars
100     FRESDWNT_DATA_RATE_TX, // 11M
101     FRESDWNT_DATA_RATE_SUPPORTED_TX, // 1M,11M,54M   126chars
102     FRESDWNT_MAC_ADDRESS,
103     FRESDWNT_MEDIUM_OCCUPANCY_LIMIT, // 0..1000
104     FRESDWNT_CONTENTION_FREE_POLLABLE, // TRUE
105     FRESDWNT_CONTENTION_FREE_PERIOD, // 0..255
106     FRESDWNT_CONTENTION_FREE_MAX_DURATION, //0..65536
107     FRESDWNT_AUTHENTICATION_RESPONSE_TIMEOUT, // 1..4294967295
108     FRESDWNT_PRIVACY_OPTION_IMPLEMENTED, // FALSE (WEP DESABILITADO)
109     FRESDWNT_POWER_MANAGEMENT_MODE, // 1-active  2-powersave
110     FRESDWNT_BEACON_PERIOD, // 1..65535
111     FRESDWNT_DTIM_PERIOD, // 1..255
112     FRESDWNT_ASSOCIATION_RESPONSE_TIMEOUT, //1..4294967295
113     FRESDWNT_DISASSOCIATE_REASON, // 1..65536
114     FRESDWNT_DISASSOCIATE_STATION, // MAC
115     FRESDWNT_DEAUTHENTICATE_REASON, // 1..65536
116     FRESDWNT_DEAUTHENTICATE_STATION, // MAC
117     FRESDWNT_AUTHENTICATE_FAIL_STATUS, // 1..65535
118     FRESDWNT_AUTHENTICATE_FAIL_STATION, // MAC
119     FRESDWNT_AUTHENTICATION_ALGORITH, // 1-OPEN SYSTEM    2-SHARED KEY
120 // CELULAR
121 };
122 enum fresdwn_type_of_frequency_atribute {
123 // 802.11
124     FRESDWNT_2_4Ghz,
125     FRESDWNT_5Ghz,
126 // CELULAR
127     FRESDWNT_850Mhz,
128     FRESDWNT_900Mhz,
129     FRESDWNT_1700Mhz,
130     FRESDWNT_1800Mhz,
131     FRESDWNT_1900Mhz,
132     FRESDWNT_2100Mhz,
133 };
134
135 enum fresdwn_action_subtype {
136     FRESDWN_ACTION_SEND,
137     FRESDWN_ACTION_RECEIVE,
```

```
138     FRESDWN_ACTION_PROCESSING
139
140 };
141
142 /* Action structure for FRESDWN_ACTION_SEND. */
143 struct fresdwn_action_send {
144     uint16_t type;                  /* OFPAT_VENDOR. */
145     uint16_t len;                   /* Length is 16. */
146     uint32_t vendor;                /* FRESDWN_VENDOR_ID. */
147     uint16_t subtype;               /* FRESDWN_ACTION_SEND. */
148     uint16_t in_port;               /* New in_port for checking flow table. */
149     uint8_t pad[4];
150 };
151 OFP_ASSERT(sizeof(struct fresdwn_action_send) == 16);
152
153 /* Action structure for FRESDWN_ACTION_RECEIVE. */
154 struct fresdwn_action_receive {
155     uint16_t type;                  /* OFPAT_VENDOR. */
156     uint16_t len;                   /* Length is 16. */
157     uint32_t vendor;                /* FRESDWN_VENDOR_ID. */
158     uint16_t subtype;               /* FRESDWN_ACTION_RECEIVE. */
159     uint16_t in_port;               /* New in_port for checking flow table. */
160     uint8_t pad[4];
161 };
162 OFP_ASSERT(sizeof(struct fresdwn_action_receive) == 16);
163
164 /* Header for FRESDWN-defined actions. */
165 struct fresdwn_action_header {
166     uint16_t type;                  /* OFPAT_VENDOR. */
167     uint16_t len;                   /* Length is 16. */
168     uint32_t vendor;                /* FRESDWN_VENDOR_ID. */
169     uint16_t subtype;               /* FRESDWN_ACTION_*. */
170     uint8_t pad[6];
171 };
172 OFP_ASSERT(sizeof(struct fresdwn_action_header) == 16);
```

Código fonte 14 – include/openflow/fresdwn-ext.h

A.6.2 include/openflow/automake.mk

```
3 include/openflow/fresdwn-ext.h \
```

Código fonte 15 – Alterações do include/openflow/automake.mk

ANEXO B – Controladora RYU

A controladora RYU é um framewrok baseado em componentes para sdn, provê uma API que permite aos desenvolvedores criar novas gerências de redes e controlar aplicações. É multiprotocolo, podendo suportar Openflow, Netconf, OF-config, dentre outros. Possui um suporte bem extenso ao Openflow, suportando plenamente as versões 1.0, 1.2, 1.3, 1.4, 1.5 além das extensões da Nicira.

A controladora RYU utilizando o código padrão de *Learning Switch*, tem um funcionamento de um switch de aprendizagem L2. Nesta aplicação, o comutador examinará cada pacote e aprenderá o mapeamento de porta de origem. Posteriormente, o endereço MAC de origem será associado à porta. Se o destino do pacote já está associado a alguma porta, o pacote será enviado para a determinada porta, senão ele será inundado em todas as demais portas do switch.

Segue abaixo código que permite a troca de mensagens experimenter para validar a proposta do framework Fresdwn.

B.1 run.py

```
1  from ryu.cmd import manager
2  import sys
3
4  def main():
5      sys.argv.append('main_app.py')
6      sys.argv.append('--verbose')
7      sys.argv.append('--enable-debugger')
8      manager.main()
9
10 if __name__ == '__main__':
11     main()
```

Código fonte 16 – run.py

B.2 main$_{a}pp.py$

```
1  import json
2  import time
3
4  from ryu.base import app_manager
5  from ryu.controller import ofp_event
6  from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
```

```
7  from ryu.controller.handler import HANDSHAKE_DISPATCHER
8  from ryu.controller.handler import set_ev_cls
9  from ryu.lib.packet import ether_types
10 from ryu.lib.packet import ethernet
11 from ryu.lib.packet import packet
12 from ryu.ofproto import ofproto_v1_3 as ofp
13 from ryu.ofproto import ofproto_v1_3
14 from ryu.ofproto import ofproto_v1_3_parser as ofp_parser
15 from ryu.topology import event, switches
16
17
18 class SimpleSwitch13(app_manager.RyuApp):
19     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
20     #OFP_VERSIONS = [ofp.OFP_VERSION, ofproto_v1_0.OFP_VERSION]
21
22     _CONTEXTS = {
23         'switches': switches.Switches
24                 }
25
26     def __init__(self, *args, **kwargs):
27         super(SimpleSwitch13, self).__init__(*args, **kwargs)
28         self.mac_to_port = {}
29         #self.__sent_packets = []
30
31     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
32     def switch_features_handler(self, ev):
33         datapath = ev.msg.datapath
34         ofproto = datapath.ofproto
35         parser = datapath.ofproto_parser
36
37         # install table-miss flow entry
38         #
39         # We specify NO BUFFER to max_len of the output action due to
40         # OVS bug. At this moment, if we specify a lesser number, e.g.,
41         # 128, OVS will send Packet-In with invalid buffer_id and
42         # truncated packet data. In that case, we cannot output packets
43         # correctly.  The bug has been fixed in OVS v2.1.0.
44         match = parser.OFPMatch()
45         actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
46                                           ofproto.OFPCML_NO_BUFFER)]
47         self.add_flow(datapath, 0, match, actions)
48
49     def add_flow(self, datapath, port, dst, actions):
50         ofproto = datapath.ofproto
51
52         match = datapath.ofproto_parser.OFPMatch(in_port=port,
53                                                  eth_dst=dst)
```

```
54          inst = [datapath.ofproto_parser.OFPInstructionActions(
55                  ofproto.OFPIT_APPLY_ACTIONS, actions)]
56
57          mod = datapath.ofproto_parser.OFPFlowMod(
58              datapath=datapath, cookie=0, cookie_mask=0, table_id=0,
59              command=ofproto.OFPFC_ADD, idle_timeout=0, hard_timeout=0,
60              priority=0, buffer_id=ofproto.OFP_NO_BUFFER,
61              out_port=ofproto.OFPP_ANY,
62              out_group=ofproto.OFPG_ANY,
63              flags=0, match=match, instructions=inst)
64          datapath.send_msg(mod)
65
66      #@set_ev_cls(event.EventLinkRequest, [MAIN_DISPATCHER, CONFIG_DISPATCHER])
67      #def _request_handler(self, ev):
68      #    pass
69
70      #@set_ev_cls(event.EventSwitchEnter, [MAIN_DISPATCHER, CONFIG_DISPATCHER])
71      #def _connectionUp_handler(self, ev):
72      #    pass
73
74      #@set_ev_cls(ofp_event.EventOFPStateChange, [MAIN_DISPATCHER, CONFIG_DISPATCHER])
75      @set_ev_cls(ofp_event.EventOFPStateChange, [CONFIG_DISPATCHER])
76      def _state_change(self, ev):
77          dp = ev.datapath
78          print("OFPStateChange {ip}:{port}".format(ip=ev.datapath.address[0],
79                                                    port=ev.datapath.address[1]))
80          actions = [ofp_parser.OFPActionOutput(ofp.OFPP_FLOOD)]
81          # out = ofp_parser.OFPPacketOut(datapath=dp, in_port=1, actions=actions)
82          data = json.dumps( msg.FRESDWN )
83          data = data + bytearray (1398 - len(data))
84          out = ofp_parser.OFPExperimenter(datapath=dp,
85                                           experimenter=0x00000005,
86                                           exp_type=20,
87                                           data=bytearray(data))
88          dp.send_msg(out)
89          time.sleep(.3)
90          #self.__sent_packets.append(out)
91
92      #@set_ev_cls(ofp_event.EventOFPHello, [MAIN_DISPATCHER, CONFIG_DISPATCHER,
        ↪  HANDSHAKE_DISPATCHER])
93      #def _hello_handler(self, ev):
94      #    pass
95          # print("OFPHello {ip}:{port}".format(ip=ev.datapath.address[0],
96          #                                    port=ev.datapath.address[1]))
97
98      #@set_ev_cls(ofp_event.EventOFPErrorMsg, [MAIN_DISPATCHER, CONFIG_DISPATCHER,
        ↪  HANDSHAKE_DISPATCHER])
```

```
99      #def _error_message_handler(self, ev):
100     #    pass
101     #
102     # @set_ev_cls(ofp_event.EventOFPEchoRequest, [MAIN_DISPATCHER, CONFIG_DISPATCHER,
        ↪  HANDSHAKE_DISPATCHER])
103     # def _echo_request_handler(self, ev):
104     #     pass
105
106     @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
107     def _packet_in_handler(self, ev):
108         msg = ev.msg
109         datapath = msg.datapath
110         ofproto = datapath.ofproto
111         in_port = msg.match['in_port']
112
113         pkt = packet.Packet(msg.data)
114         eth = pkt.get_protocols(ethernet.ethernet)[0]
115
116         if eth.ethertype == ether_types.ETH_TYPE_LLDP:
117             # ignore lldp packet
118             return
119         dst = eth.dst
120         src = eth.src
121
122         dpid = datapath.id
123         self.mac_to_port.setdefault(dpid, {})
124
125         self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
126
127         # learn a mac address to avoid FLOOD next time.
128         self.mac_to_port[dpid][src] = in_port
129
130         if dst in self.mac_to_port[dpid]:
131             out_port = self.mac_to_port[dpid][dst]
132         else:
133             out_port = ofproto.OFPP_FLOOD
134
135         actions = [datapath.ofproto_parser.OFPActionOutput(out_port)]
136
137         # install a flow to avoid packet_in next time
138         if out_port != ofproto.OFPP_FLOOD:
139             self.add_flow(datapath, in_port, dst, actions)
140
141         data = None
142         if msg.buffer_id == ofproto.OFP_NO_BUFFER:
143             data = msg.data
144
```

```
145         out = datapath.ofproto_parser.OFPPacketOut(
146             datapath=datapath, buffer_id=msg.buffer_id, in_port=in_port,
147             actions=actions, data=data)
148         datapath.send_msg(out)
```

Código fonte 17 – main$_app.py$

ANEXO C – Ambiente integrado Mininet/NS-3

É desmembrado do projeto Opennet que se classifica como um simulador para Redes sem fio definidas por software (Software-Defined Wireless Local Area Network - SDWLAN) construído sobre o Mininet e o ns-3.
Neste código abaixo foi alterado o switch padrão de ovs para o ofsoftswitch13 customizado conforme o anexo A, portanto ao estabelecer uma comunicação com a controladora, ocorre a troca daas mensagens experimente3r, o que permite a rápida configuração e customização do ambiente.

C.1 wifi$_m$*obility.py*

```
#!/usr/bin/python

"""
This example shows how to create an empty Mininet object
(without a topology object) and add nodes to it manually.
"""
import sys
import os

import mininet.net
import mininet.node
import mininet.cli
import mininet.log
import mininet.ns3

from mininet.net import Mininet, MininetWithControlNet
from mininet.node import Controller, RemoteController, UserSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.ns3 import *

import ns.core
import ns.network
import ns.wifi
import ns.csma
import ns.wimax
import ns.uan
import ns.netanim
```

```
29
30 from mininet.opennet import *
31
32 """
33 nodes is a list of node descriptions, each description contains several node
   ↪ attributes.
34
35 name: The node identifier.
36 type: The character of this node, can be host or switch.
37
38 position: The position of this node.
39 velocity: The velocity of this node, which will be used in host mobility model.
40           The default mobility model is ns3::ConstantVelocityMobilityModel.
41
42 mobility: The mobility model of this node. setListPoritionAllocate() will take two
   ↪ parameters,
43           MobilityHelper and ListPositionAllocate. After install the PositionAllocate
   ↪ in the
44           MobilityHelper, setListoritionAllocate() will return the MobilityHelper.
45 """
46
47 nodes = [ { 'name': 'h1', 'type': 'host', 'ip': '10.10.10.1', 'position': (0.0, 10.0,
   ↪ 0.0), 'velocity': (2.5, 0, 0) },
48           { 'name': 'h2', 'type': 'host', 'ip': '10.10.10.2', 'mobility':
            ↪ setListPositionAllocate(
49              createMobilityHelper("ns3::RandomWalk2dMobilityModel",n0="Bounds",
50              v0=ns.mobility.RectangleValue(ns.mobility.Rectangle(100,200,-50,50))),
51              createListPositionAllocate(x1=150,y1=30,z1=0)) },
52           { 'name': 's1', 'type': 'switch', 'position': (0.0, 0.0, 0.0) },
53           { 'name': 's2', 'type': 'switch', 'position': (120.0, 0.0, 0.0) },
54           { 'name': 's3', 'type': 'switch', 'position': (60.0, 60.0*(3**0.5), 0.0) },
55           { 'name': 's4', 'type': 'switch', 'position': (60.0, -60.0*(3**0.5), 0.0) },
56           { 'name': 's5', 'type': 'switch', 'position': (-120.0, 0.0, 0.0) },
57           { 'name': 's6', 'type': 'switch', 'position': (-60.0, 60.0*(3**0.5), 0.0) },
58           { 'name': 's7', 'type': 'switch', 'position': (-60.0, -60.0*(3**0.5), 0.0)
            ↪ },
59         ]
60
61 """
62 wifiintfs is a list of wifi interface descriptions, each description contains some
   ↪ wifi attributes.
63 """
64
65 wifiintfs = [ {'nodename': 'h1', 'type': 'sta', 'channel': 1, 'ssid': 'ssid'},
66               {'nodename': 'h2', 'type': 'sta', 'channel': 11, 'ssid': 'ssid'},
67               {'nodename': 's1', 'type': 'ap', 'channel': 1, 'ssid': 'ssid'},
68               {'nodename': 's2', 'type': 'ap', 'channel': 6, 'ssid': 'ssid'},
```

```
69              {'nodename': 's3', 'type': 'ap', 'channel': 11, 'ssid': 'ssid'},
70              {'nodename': 's4', 'type': 'ap', 'channel': 11, 'ssid': 'ssid'},
71              {'nodename': 's5', 'type': 'ap', 'channel': 6, 'ssid': 'ssid'},
72              {'nodename': 's6', 'type': 'ap', 'channel': 11, 'ssid': 'ssid'},
73              {'nodename': 's7', 'type': 'ap', 'channel': 11, 'ssid': 'ssid'},
74          ]
75
76  """
77  links is a list of Ethernet links.
78  """
79
80  links = [ {'nodename1': 's1', 'nodename2': 's2'},
81            {'nodename1': 's1', 'nodename2': 's3'},
82            {'nodename1': 's1', 'nodename2': 's4'},
83            {'nodename1': 's1', 'nodename2': 's5'},
84            {'nodename1': 's1', 'nodename2': 's6'},
85            {'nodename1': 's1', 'nodename2': 's7'},
86        ]
87
88  def WifiNet():
89
90      """ Create an Wifi network and add nodes to it. """
91
92      net = Mininet(switch=UserSwitch)
93
94      info( '*** Adding controller\n' )
95      net.addController( 'c0', controller=RemoteController, ip='127.0.0.1', port=6633 )
96
97      """ Initialize the WifiSegment, please refer ns3.py """
98      wifi = WifiSegment(standard = ns.wifi.WIFI_PHY_STANDARD_80211g)
99      wifinodes = []
100
101     """ Initialize nodes """
102     for n in nodes:
103
104         """ Get attributes """
105         nodename = n.get('name', None)
106         nodetype = n.get('type', None)
107         nodemob = n.get('mobility', None)
108         nodepos = n.get('position', None)
109         nodevel = n.get('velocity', None)
110         nodeip = n.get('ip', None)
111
112         """ Assign the addfunc, please refer Mininet for more details about addHost
            ↪  and addSwitch """
113         if nodetype is 'host':
114             addfunc = net.addHost
```

```
115         elif nodetype is 'switch':
116             addfunc = net.addSwitch
117         else:
118             addfunc = None
119         if nodename is None or addfunc is None:
120             continue
121 
122         """ Add the node into Mininet """
123         node = addfunc (nodename, ip=nodeip)
124 
125         """ Set the mobility model """
126         mininet.ns3.setMobilityModel (node, nodemob)
127         if nodepos is not None:
128             mininet.ns3.setPosition (node, nodepos[0], nodepos[1], nodepos[2])
129         if nodevel is not None:
130             mininet.ns3.setVelocity (node, nodevel[0], nodevel[1], nodevel[2])
131 
132         """ Append the node into wifinodes """
133         wifinodes.append (node)
134 
135     """ Initialize Wifi Interfaces """
136     for wi in wifiintfs:
137 
138         """ Get attributes """
139         winodename = wi.get('nodename', None)
140         witype = wi.get('type', None)
141         wichannel = wi.get('channel', None)
142         wissid = wi.get('ssid', None)
143 
144         """ Assign the addfunc, please refer the WifiSegment in ns3.py """
145         if witype is 'sta':
146             addfunc = wifi.addSta
147         elif witype is 'ap':
148             addfunc = wifi.addAp
149         else:
150             addfunc = None
151         if winodename is None or addfunc is None or wichannel is None:
152             continue
153 
154         """ Get wifi node and add it to the TapBridge """
155         node = getWifiNode (wifinodes, winodename)
156         addfunc (node, wichannel, wissid)
157 
158     """ Initialize Ehternet links between switches """
159     for cl in links:
160         clnodename1 = cl.get('nodename1', None)
161         clnodename2 = cl.get('nodename2', None)
```

```
162         if clnodename1 is None or clnodename2 is None:
163             continue
164         clnode1 = getWifiNode (wifinodes, clnodename1)
165         clnode2 = getWifiNode (wifinodes, clnodename2)
166         if clnode1 is None or clnode2 is None:
167             continue
168         net.addLink( clnode1, clnode2 )
169 
170     """ Enable Pcap output"""
171     pcap = Pcap()
172     pcap.enable()
173     print pcap
174 
175     """ Enable netanim output"""
176     anim = Netanim("/tmp/xml/wifi-wired-bridged4.xml", nodes)
177     print anim
178 
179     """ Update node descriptions in the netanim """
180     for node in wifinodes:
181         anim.UpdateNodeDescription (node.nsNode, str(node) + '-' +
        ↪  str(node.nsNode.GetId()))
182         if isinstance(node, mininet.node.OVSSwitch):
183             color = (0, 255, 0)
184         elif isinstance(node, mininet.node.Host):
185             color = (0, 0, 255)
186         anim.UpdateNodeColor (node.nsNode, color[0], color[1], color[2])
187 
188     """ Start the simulation """
189     info( '*** Starting network\n' )
190     net.start()
191     mininet.ns3.start()
192 
193     info( 'Testing network connectivity\n' )
194     wifinodes[0].cmdPrint( 'ping 10.10.10.2 -c 3' )
195 
196     CLI( net )
197 
198     info( '*** Stopping network\n' )
199     mininet.ns3.stop()
200     info( '*** mininet.ns3.stop()\n' )
201     mininet.ns3.clear()
202     info( '*** mininet.ns3.clear()\n' )
203     net.stop()
204     info( '*** net.stop()\n' )
205 
206 if __name__ == '__main__':
207     setLogLevel( 'info' )
```

```
208     WifiNet()
```

Código fonte 18 – wifi$_m$*obility.py*