ETHANOL: UMA PLATAFORMA SDN PARA REDES WI-FI

HENRIQUE DUARTE MOURA

ETHANOL: UMA PLATAFORMA SDN PARA REDES WI-FI

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Daniel Fernandes Macedo
Coorientador: Marcos Augusto Menezes Vieira

Belo Horizonte - MG

Julho de 2015

Moura, Henrique Duarte

M929e Ethanol: uma plataforma SDN para redes Wi-Fi / Henrique Duarte Moura. — Belo Horizonte - MG, 2015
xxix, 132 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação

Orientador: Daniel Fernandes Macedo
Coorientador: Marcos Augusto Menezes Vieira

1. Computação - Teses. 2. Redes de computadores - Teses. 3. Redes sem fio - Teses. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*22(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Ethanol: uma plataforma SDN para redes Wi-fi

HENRIQUE DUARTE MOURA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. DANIEL FERNANDES MACEDO - Orientador
Departamento de Ciência da Computação - UFMG

PROF. MARCOS AUGUSTO MENEZES VIEIRA - Coorientador
Departamento de Ciência da Computação - UFMG

PROF. CHRISTIAN RODOLFO ESTEVE ROTHENBERG
Departamento de Engenharia de Computação e Automação Industrial - UNICAMP

PROF. ÍTALO FERNANDO SCOTÁ CUNHA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 10 de julho de 2015.

Dedico este trabalho aos meus pais e à Ana Paula, pela paciência e incentivo.

Agradecimentos

Aos meus orientadores Daniel e Marcos, pelo auxílio e orientação inestimáveis.
À Ana Paula e meus familiares, por tudo mais.
Aos colegas do laboratório Winet, pelo apoio e ideias, e em especial ao Gabriel.
Aos amigos, por suportarem toda a minha ausência.
Aos professores do PPGCC, pelos conhecimentos e orientação.
Aos funcionários da secretaria do DCC, por resolverem sempre os problemas.
Às agencias financiadoras do projeto de pesquisa - CAPES e CNPq, que viabilizaram minha dedicação aos estudos.

"The known is finite, the unknown infinite; intellectually we stand on an islet in the midst of an illimitable ocean of inexplicability. Our business in every generation is to reclaim a little more land."

(T. H. Huxley, 1887)

Resumo

As redes sem fio estão presentes em quase todos os ambientes empresariais, e, também, nas residências com acesso Internet de banda larga. Para suportar a crescente demanda dos usuários móveis, notamos uma tendência para redução da área de cobertura do sinal de rede sem fio de um ponto de acesso e ao mesmo tempo o aumento na quantidade destes dispositivos. Diversas soluções proprietárias procuram resolver os problemas decorrentes do adensamento dos pontos de acesso, como maior interferência e maior quantidade de troca das estações entre antenas. Isso faz com que a introdução de qualquer novo dispositivo de rede, a manutenção ou correção de problemas em um dispositivo existente seja um trabalho tedioso e sujeito a erros, porque requer a reconfiguração de cada um dos numerosos dispositivos de rede. O gerenciamento dos dispositivos de rede é penoso para os administradores de rede, que sofrem pela falta de interoperabilidade entre as soluções dos diversos fabricantes e por não conseguirem soluções sob medida para suas necessidades.

Software-Defined Networking (SDN) é uma abordagem para redes de computadores que permite gerenciá-las por meio de uma abstração na qual há o desacoplamento do sistema que toma decisões sobre o fluxo de dados (plano de controle) dos sistemas subjacentes que encaminham os dados (o plano de dados). As soluções SDNs atuais, que utilizam principalmente o protocolo OpenFlow, são insuficientes para tratar o dinamismo de uma rede sem fio, suas necessidades de segurança e os aspectos de qualidade de serviço (QoS). O protocolo OpenFlow não implementa primitivas capazes de atuar sobre os aspectos de configuração e gerenciamento. Para melhorar o desempenho dessas redes são necessárias novas abordagens, que não são possíveis em controladoras WLAN (*Wireless Local Area Networks*) proprietários e fechados. Uma solução é a utilização de API (*Application Programming Interfaces*) abertas que permitam o desenvolvimento de aplicações e algoritmos de controle sensíveis ao contexto.

Nesta dissertação propomos uma arquitetura SDN para redes sem fio IEEE 802.11, denominada *Ethanol*, cuja API permite que sejam desenvolvidos softwares de controle customizados. O *Ethanol* refatora as funcionalidades do plano de controle

e de dados entre os pontos de acesso sem fio e o controlador SDN, sem a necessidade de alteração dos clientes sem fio[1]. Este controlador centralizado possui uma visão global da rede. O controlador pode gerenciar os pontos de acesso sem fio que receberam o agente *Ethanol*. Isto permite, por exemplo, que controlador crie filas de prioridade de encaminhamento, atuando sobre a qualidade de serviço nos enlaces. Pode ainda modificar outros parâmetros dos enlaces sem fio. Desta forma, o controlador pode, por exemplo, gerenciar o *handoff* dos dispositivos e o processo de associação e desassociação de estações sem fio.

Utilizando a arquitetura proposta foi criado um protótipo de um ponto de acesso *Ethanol*, sendo seu funcionamento e desempenho avaliado em quatro estudos de caso envolvendo a qualidade do serviço, o controle de associação cliente, a supressão de pacotes ARP e a identificação de problemas na interface sem fio de pontos de acesso.

Os resultados obtidos nos estudos de caso mostram 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 densas. Como trabalhos futuros, sugerimos estender o *Ethanol* com os protocolos propostos pelo IEEE para auxiliar o funcionamento de redes 802.11, quando estes protocolos estiverem implementados nos *drivers* 802.11.

Palavras-chave: Redes de computadores, Redes sem fio, IEEE 802.11, Redes definidas por software, SDN, Redes sem fio definidas por software, SDWN.

[1]Diversos termos são utilizados para referenciar genericamente os dispositivos em uma rede sem fio. Podemos identificar os termos estações, nós, nodos, dispositivos sem fio, clientes sem fio etc. Nesta dissertação utilizaremos os termos dispositivo sem fio e estação sem fio de forma indistinta.

Abstract

Wireless networks are common in most enterprise environments, as well as in homes with broadband Internet access. To support mobile users' growing demand, we noticed a tendency to reduce the coverage area of an access point's wireless signal while the number of these devices increases.

Several proprietary solutions seek to solve the problems arising from the densification of access points, such as increased interference and greater amount of stations roaming among acess points changing antennas. The introduction of a new network device, the device's maintenance or correcting problems in an existing device makes the administrator work tedious and error prone, because that requires the reconfiguration of each of the numerous network nodes. The management of network stations is painful for network administrators, which suffer from a lack of interoperability between different manufacturers' solutions and fail to obtain solutions tailored to their needs.

Software-defined networking (SDN) is an approach to computer networks that allows us to manage it through an abstraction in which the system that makes decisions about the data flow (control plane) is decoupled from the underlying systems that forward data (data plane).

Current SDN solutions, that use the OpenFlow protocol, are insufficient to deal with the dynamicity of wireless networks, their security needs and their aspects of quality of service (QoS). To improve the performance of these networks requires new approaches, which are not possible in commercial WLAN (Wireless local area networks) controllers, since they are proprietary and closed. One solution is to use open APIs that enable the development of applications and context sensitive control algorithms.

In this Master's thesis we propose an SDN architecture for wireless networks based on IEEE 802.11, called *Ethanol*, whose API allows the development of custom control software. *Ethanol* refactors the control and data plane functionalities between wireless access points and the SDN controller without changing the wireless stations. This centralized controller, which has a global view of the network, operates the wireless APs, controlling the quality of service in the links, client mobility and the process

of association and disassociation of wireless stations as well as other wireless links parameters.

Using the *Ethanol* architecture we create an *Ethanol* access point prototype, whose operation and performance were evaluated on four case studies involving quality of service, client association control, suppression of ARP packets and identification of problems at the wireless interfaces. Our case studies' results show that the approach of software defined wireless networks is feasible, and SDN allows to act upon various management problems of dense wireless networks.

As future work, we propose to extend *Ethanol* to comply with other IEEE protocols proposed to assist the operation of 802.11 networks when these protocols are implemented in the 802.11 drivers, and also test it under a mesh network.

Keywords: Computer Networks, Wireless networks, IEEE 802.11, Software defined networks, SDN, Software defined wireless networks, SDWN.

Lista de Figuras

2.1 Arquitetura das Redes Definidas por Software 9
2.2 Arquitetura básica do OpenFlow 12
2.3 Fluxograma detalhando o fluxo de pacotes em um comutador OpenFlow a partir da versão 1.3.0 13

4.1 Arquitetura da solução com *Ethanol* 44
4.2 Esquema de comunicação cliente servidor utilizando XML-RPC 47
4.3 Modelo da API do *Ethanol* 51
4.4 Processo de associação simplificado 54

5.1 Processo de transação para o protocolo de resolução de endereço 64
5.2 Esquema de conexão para experimento de controle de fluxo ARP 67
5.3 Vazão no cenário de controle de fluxo ARP 68
5.4 Distribuição cumulativa de frequências dos RTT 71
5.5 Esquema de conexão para experimento de controle de fluxo 73
5.6 Tráfego gerado pelas 3 estações com filas de prioridade no roteador *Ethanol* 74
5.7 Tráfego gerado com as 3 estações no roteador *Ethanol* sem controle de QoS 75
5.8 Distribuição acumulativa de frequências para RTT 76
5.9 Controle do processo de associação de uma estação à rede sem fio realizado pelo *Ethanol* 78
5.10 Esquema de conexão para experimento de controle de associação 79
5.11 Número de estações ativas por ponto de acesso 80
5.12 Distribuição acumulativa de frequências para os tempos de processamento 82
5.13 Esquema de conexão para experimento de verificação de interfaces sem fio 85
5.14 Detecção de pontos de acesso sem fio por roteadores *Ethanol* 86
5.15 Detecção de falhas em interface sem fio em roteadores *Ethanol* 87
5.16 Distribuição cumulativa dos tempo de processamento de *beacons* 87
5.17 Linha de tempo para detecção de falhas 89

5.18 Uso de recursos no computador que executará o controlador 90
5.19 Comparação de uso relativo de memória e processador nos estudos de casos do controlador 100
5.20 Comparação de uso relativo de memória e processador nos estudos de casos do roteador 101

A.1 Frequências sobrepostas em 2.4GHz 108
A.2 Frequências em 5GHz nos Estados Unidos da América 109
A.3 Esquema comparativo entre transmissões por rajadas ou sem rajadas . . . 116
A.4 Processo de associação de uma estação à rede sem fio 117

Lista de Tabelas

3.1 Família do padrão IEEE 802.11 21
3.2 Comparação entre as propostas 41
3.3 Comparação entre as controladoras comerciais 42

4.1 Acréscimo de tamanho ao módulo *hostapd* 50

5.1 Processamento de pacotes ARP pelo controlador em número de pacotes . . 69
5.2 Tempo de processamento de pacotes ARP pelo controlador (em microsegundos) 70
5.3 Tempo de processamento de pacotes pelo controlador (em microsegundos) 75
5.4 Tempo de processamento de pacotes pelo controlador durante processo de associação (em microssegundos) 81
5.5 Verificação dos efeitos dos parâmetros de configuração sobre o tempo de detecção de falhas pelo controlador 88
5.6 Uso de memória no computador com controlador desativado 91
5.7 Uso de CPU no computador com controlador desativado 91
5.8 Uso de CPU no controlador durante o experimento de controle ARP 91
5.9 Uso de memória no controlador durante o experimento de controle ARP . 92
5.10 Uso de memória no controlador durante experimento de controle de fluxos 92
5.11 Uso de CPU no controlador durante experimento de controle de fluxos . . 92
5.12 Uso de memória no controlador durante experimento de controle de associação 93
5.13 Uso de CPU no controlador durante experimento de controle de associação 93
5.14 Uso de memória no controlador durante experimento de identificação de falhas na interface sem fio 94
5.15 Uso de CPU no controlador durante experimento de identificação de falhas na interface sem fio 94
5.16 Uso de memória no computador com roteador desativado 95
5.17 Uso de CPU no computador com roteador desativado 95

5.18 Uso de processador no roteador durante o experimento de controle ARP . 96
5.19 Uso de memória no roteador durante o experimento de controle ARP . . . 96
5.20 Uso de memória no roteador durante experimento de controle de fluxos . . 96
5.21 Uso de CPU no roteador durante experimento de controle de fluxos 97
5.22 Uso de memória no roteador durante experimento de controle de associação 97
5.23 Uso de CPU no roteador durante experimento de controle de associação . . 98
5.24 Uso de memória no roteador durante experimento de identificação de falhas na interface sem fio 98
5.25 Uso de CPU no roteador durante experimento de identificação de falhas na interface sem fio 99

A.1 Taxas de transmissão em função da largura do canal 109

Lista de Algoritmos

1 Controle de transmissão de ARP na rede 65
2 Controle de fluxo de dados utilizando fila de prioridade 72
3 Controle do processo de associação de uma estação a um AP 77
4 Identificação de falha na interface sem fio de pontos de acesso 83

Lista de Símbolos

3GPP *3rd Generation Partnership Project*

AAA *Authentication, Authorization and Accounting*

AIFS *Arbitration Inter-Frame Spacing*

API *Application Programming Interface*

APs *Access Points* ou Pontos de acesso sem fio

ARP *Address Resolution Protocol*

ASICs *Application-specific integrated circuits*

BSS *Basic Service Set*

BSSID *Basic Service Set Identification*

CAPEX *Capital Expenditure*

CAPWAP *Control And Provisioning of Wireless Access Points*

CoAP *Coordination framework for Open APs*

CPU *Central Processing Unit*

CROWD *Connectivity management for eneRgy Optimised Wireless Dense networks*

DAIR *Dense Array of Inexpensive Radios*

dBm decibel miliwatt, equivale a $P_{dBm} = 10log_{10}\left(\frac{P_{mW}}{1mW}\right)$

DHCP *Dynamic Host Configuration Protocol*

DPI *Deep Packet Inspection*

DTIM *Delivery Traffic Indication Message*

EDCA *Enhanced Distributed Channel Access*

ESS *Extended Service Set*

ESSID *Extended Service Set Identification*

FPGAs *Field Programmable Gate Arrays*

FSPL *Free Space Loss*

GATS *Group Addressed Transmission Service*

GPS *Global Positioning System*

HCCA HCF *Controlled Channel Access*

HCF *Hybrid Coordination Function*

HTB *Hierarchical Token Bucket*

HTTP *Hypertext Transfer Protocol*

HTTPS *Hyper Text Transfer Protocol Secure*

IaaS *Infrastructure as a Service*

IAPP *Inter-Access Point Protocol*

IBSS *Independent Basic Service Set*

IEEE *Institute of Electrical and Electronics Engineers*

IETF *Internet Engineering Task Force*

IP *Internet Protocol*

ISI *Intersymbol Interference*

ISM *Industrial, Scientific and Medical radio bands*

LAG *Link Aggregation*

LTE *Long Term Evolution*

MIMO *Multiple-Input and Multiple-Output*

NetConf *Network configuration* é um protocolo para configuração de elementos de redes

NFP *Network flow processors* - processadores de fluxo de rede

ONF *Open Networking Foundation*

OPEX *Operational Expenditures*

P2P *Peer to Peer*

P2P-GO P2P *Group Owner*

PLDs *Programmable logic devices*

QMF *Quality-of-service Management Frame*

QoE *Quality of experience*

QoS *Quality of Service*

RADIUS *Remote Authentication Dial-In User Service*

RAM *Random Access Memory*

RAN *Radio Access Network*

RF radiofrequência

RFC *Request for Comments*

RPC *Remote Procedure Call*

RSS *Received Signal Strength*

RTS sinal de *Request To Send*

RTT *Round Trip Time*

SDK *Software Development Kit*

SDN *Software Defined Network*

SDWN *Software Defined Wireless Network*

SLA *Service Level Agreement*

SNIR Signal to Noise plus Interference Ratio

SNMP *Simple Network Management Protocol*

SNR *Signal-to-Noise Ratio*

SSID *Service Set IDentification*

TCP *Transfer Control Protocol*

TSF *Timing Synchronization Function*

USB *Universal Serial Bus*

VAP *Virtual Access Point*

VM *Virtual Machines*

VoWLAN *Voice over Wireless* LAN

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

WiMax *Worldwide Interoperability for Microwave Access*/Interoperabilidade Mundial para Acesso de Micro-Ondas

WIPS *Wireless Intrusion Detection System*

WLAN *Wireless Local Area Network*

WMN *Wireless Mesh Networks*

XML *Extensible Markup Language*

Sumário

Agradecimentos **ix**

Resumo **xiii**

Abstract **xv**

Lista de Figuras **xvii**

Lista de Tabelas **xix**

1 Introdução **1**
1.1 Problema 4
1.2 Objetivos e Metas 4
1.3 Resultados atingidos 5
1.4 Organização da dissertação 5

2 Redes Definidas por Software **7**
2.1 A abordagem de Redes Definidas por Software 8
2.2 O Protocolo OpenFlow 11
2.2.1 Controladores 13
2.3 Limitações de SDN 14
2.4 Conclusões 16

3 Gerenciamento de redes Wi-Fi **17**
3.1 Gerenciamento Tradicional 18
3.1.1 Controladoras proprietárias para redes Wi-Fi 18
3.1.2 Protocolos de gerenciamento e controle de redes Wi-Fi 20
3.2 Desafios em redes sem fio definidas por software 25
3.2.1 Características variáveis dos enlaces 26
3.2.2 Mobilidade 27

3.2.3 Qualidade de serviço 28
3.2.4 Virtualização 29
3.2.5 Segurança 29
3.2.6 Localização 30
3.2.7 Configuração 31
3.3 Trabalhos relacionados de gerenciamento SDN 32
3.3.1 Redes IEEE 802.11 33
3.3.2 Redes celulares 35
3.3.3 Monitoramento de redes sem fio 37
3.4 Classificação do gerenciamento baseado em SDN 38
3.5 Conclusão 40

4 Ethanol: Uma plataforma SDN para redes Wi-Fi 43
4.1 Arquitetura do *Ethanol* 44
4.2 Implementação do *Ethanol* 45
4.2.1 Plataformas de hardware do *Ethanol* 47
4.2.2 Roteador *Ethanol* empregando XML-RPC 48
4.3 Operações suportadas pelo *Ethanol* 50
4.3.1 *AccessPoint* 52
4.3.2 *Radio* 53
4.3.3 *VirtualAcesssPoint* 53
4.3.4 *Network* 55
4.3.5 *Station* 55
4.3.6 *Flow* 56
4.4 Exemplo de uso da arquitetura 56
4.5 Casos de uso do *Ethanol* 58
4.5.1 Características variáveis dos enlaces 59
4.5.2 Mobilidade 59
4.5.3 Qualidade de serviço 59
4.5.4 Segurança 60
4.5.5 Localização de usuários 61
4.5.6 Configuração 61
4.6 Conclusão 62

5 Avaliação experimental do protótipo 63
5.1 Controle de transmissões ARP 63
5.1.1 Configuração do experimento 67

5.1.2 Resultados 68
5.2 Implementação de *QoS* para tráfego Ethernet e sem fio 71
5.2.1 Configuração do experimento 73
5.2.2 Resultados 74
5.3 Controle do processo de associação de estações sem fio 77
5.3.1 Configuração do experimento 78
5.3.2 Resultados 79
5.4 Identificação de interface sem fio falha 82
5.4.1 Configuração do experimento 84
5.4.2 Resultados 85
5.5 Uso de memória e CPU nos estudos de caso 89
5.5.1 Uso de memória e processador pelo controlador *Ethanol* 90
5.5.2 Uso de memória e processador pelo roteador *Ethanol* 94
5.5.3 Comparação 99
5.6 Conclusão 100

6 Conclusões e Trabalhos Futuros **103**
6.1 Trabalhos Futuros 104

Apêndice A Descrição detalhada da API *Ethanol* **107**
A.1 *Radio* 108
A.2 AccessPoint 110
A.3 *Network* 112
A.4 *Station* 113
A.5 *VirtualAccessPoint* 115
A.6 *Flow* 118

Referências Bibliográficas **121**

Capítulo 1

Introdução

Historicamente, as redes de computadores utilizam dispositivos de encaminhamento com hardware de propósito específico, que permite a comunicação de dados com alto desempenho, mas que impõe grandes dificuldades e custos aos usuários e empresas (Foundation [2012]). Estes equipamentos são utilizados pelos administradores de rede na montagem de redes para aplicações críticas com topologias complexas e, muitas vezes, com requisitos conflitantes. A configuração dos equipamentos de rede e sua instalação requerem técnicos especializados e altamente treinados, mesmo assim obtêm-se, muitas vezes, desempenho moderado em função da complexidade da tarefa (Sezer et al. [2013]). Além disto, estes técnicos devem lidar com os custos operacionais envolvidos no provisionamento e no gerenciamento de suas redes, com múltiplos fornecedores e com tecnologias diversas, bem como procurar sempre reduzir o valor das despesas operacionais - OPEX (*Operational Expenditures*)[1]. E mais, a redução de recursos humanos especializados e o aumento dos custos crescentes geram um interesse da comunidade em soluções que possam unificar o gerenciamento de rede e seu provisionamento em vários domínios. Portanto como as redes IP tradicionais são complexas e díficeis de gerenciar (Kreutz et al. [2014]), um novo modelo de rede é necessário. Em auxílio a estes administradores, surgem iniciativas como as Redes Definidas por Software (SDN) (Casado et al. [2007, 2012]; Guedes et al. [2012]; Feamster et al. [2013]).

A Open Networking Foundation (ONF) define que, na arquitetura SDN, "os planos de controle e de dados são dissociados, a inteligência de rede e seu estado são logicamente centralizados e a infraestrutura de rede subjacente é exposta para as aplicações" (Foundation [2012]). A arquitetura SDN é formada por APIs que fornecem uma interface entre o plano de controle programável e o plano de dados, bem como

[1]Representa o montante de dinheiro utilizado por uma empresa para manter em operação seus bens de capital, como os equipamentos e instalações.

uma interface entre as aplicações/serviços e o plano de controle. (Shin et al. [2012]). Entendemos que, ao oferecer uma interface programática, a arquitetura SDN permite tratar uma ampla variedade de requisitos de operação, sem alterar quaisquer dos aspectos de nível mais baixo da rede, tornando mais simples o seu controle, uma vez que um controlador centralizado passa a possuir uma visão global. A arquitetura SDN desacopla o plano de controle do plano de dados (equipamento de rede, como comutadores e roteadores) (Fernandez [2013]). Com isto, temos, nas redes SDN, um controlador capaz de implementar políticas de controle distribuídas para os diversos comutadores com um mecanismo de supervisão centralizado.

Esta abordagem abre caminho para novos serviços de rede, como a criação de redes virtuais, a autenticação, a implementação de algoritmos de monitoramento, o acompanhamento de tráfego mais sofisticado e muitas outras aplicações. Desta forma pode-se concluir que a utilização de redes programáveis de computadores, como no paradigma SDN, resulta em aumento da inovação e reduz as barreiras para implementação de novos serviços de rede (Sezer et al. [2013]). A abordagem SDN reduz a dependência do administrador a um mesmo fornecedor de equipamentos. Esta abordagem torna ainda os algoritmos de controle mais transparentes, pois são acessíveis aos administradores que podem modificá-los de acordo com sua necessidade.

Contudo, as redes de computadores não são compostas somente pelas redes cabeadas. Encontramos redes sem fio em *campi* e nas empresas, que estão se tornando cada vez mais presentes, com cobertura mais abrangente e com demandas de tráfego crescente. Além disso, os novos padrões de comunicação sem fio, tais como o IEEE 802.11ac e o IEEE 802.11ad, requerem cada vez mais pontos de acesso e antenas com cobertura reduzida (Verma et al. [2013]; IEEE802.11ac [2013]; IEEE802.11ad [2012]; Perahia et al. [2010]), devido à necessidade de uma maior vazão da rede. Estes protocolos utilizam sinais que propagam empregando um menor comprimento de onda, como no IEEE 802.11ad, reduzindo seu raio de atuação e portanto aumentando a quantidade de dispositivos para prover a mesma cobertura. Configurar eficazmente os parâmetros dos pontos de acesso (APs) pode melhorar significativamente a capacidade das redes WLAN (*Wireless Local Area Network*) e o seu desempenho (Bhaumik et al. [2012]; Kumar et al. [2013b]; Gudipati et al. [2013]).

As abordagens atuais para gerenciamento da rede sem fio consistem tipicamente em atribuir estaticamente valores para os parâmetros de configuração nos pontos de acesso de baixo custo. Estes equipamentos possuem algoritmos simples para seleção de canal e potência de transmissão. Os usuários podem ainda utilizar equipamentos e softwares de gerenciamento proprietários (e extremamente caros) que já possuam algoritmos de controle capazes de melhorar estes parâmetros (Aruba Networks [2015];

Cisco Systems [2015a]; Meraki Networks [2015]; Dell [2015]; Extreme Networks [2015]; HP Networks [2015]). Estes algoritmos são genéricos, não atendendo a demandas específicas dos administradores de rede. Estes algoritmos são inflexíveis sob a ótica dos administradores, pois dependem dos fabricantes para realização de *patches* e atualizações.

Desta forma, os modelos atuais de gerenciamento de redes sem fio não conseguem atender plenamente as demandas dos administradores e usuários das redes sem fio. Por exemplo, os modelos atuais não permitem a adoção de soluções customizadas para um problema ou necessidade de um usuário. A aplicação dos conceitos de SDNs em redes sem fio cria a possibilidade de uma nova abordagem de gerenciamento, que permite a execução de processos do plano de controle em um controlador capaz de ter uma visão global da rede, chegando a melhores decisões para o funcionamento geral da rede.

Nesta dissertação apresentamos uma arquitetura para redes sem fio, utilizando uma abordagem SDN, que permite o gerenciamento de redes sem fio. Com esta arquitetura, que denominamos *Ethanol*, buscamos resolver problemas de gerenciamento de redes sem fio e permitir, utilizando uma API aberta, que possamos implementar novos serviços para as redes sem fio. Concentramos nosso foco nas redes Wi-Fi que suportam o protocolo IEEE 802.11. Criamos um protótipo utilizando esta arquitetura que avaliamos em quatro estudos de caso relacionados no capítulo 5. Os resultados obtidos mostram que uma abordagem de redes sem fio definidas por software é factível.

O *Ethanol* fornece uma API aberta, capaz de suportar as redes locais sem fio definidas pelo padrão IEEE 802.11, bem como as redes Ethernet. Com o *Ethanol*, fornecemos mecanismos para criação de algoritmos de controle e gerenciamento de redes. Estas aplicações consomem os serviços fornecidos pela nossa arquitetura. Usando o *Ethanol* podemos obter informações sobre o estado da rede, especificar o comportamento desejado mediante regras implementadas pelo controlador e disseminar novas configurações para os elementos de rede, visando prover o comportamento desejado. O *Ethanol* não exige mudanças nas estações. Com isto a arquitetura provê flexibilidade para o administrador de rede, fornecendo APIs para a mobilidade dos nós, para a virtualização dos pontos de acesso, para a segurança em rede sem fio e para *QoS*. Nossos estudos de caso indicam que a arquitetura proposta permite ao administrador de rede atuar sobre os diversos problemas de gerenciamento de redes Wi-Fi e Ethernet.

1.1 Problema

O problema que nos motivou decorre da percepção que "os modelos atuais de gerenciamento de redes Wi-Fi não conseguem atender as demandas dos administradores e dos usuários destas redes".

1.2 Objetivos e Metas

O objetivo da dissertação é propor e implementar uma arquitetura SDN para os pontos de acesso sem fio, chamada *Ethanol*. Com isto esperamos que seja possível implementar novos serviços sobre redes IEEE 802.11, capazes de se adaptarem dinamicamente aos requisitos de comunicação de usuários móveis, utilizando hardware de baixo custo.

A dissertação descreve uma arquitetura que permite rodar programas no plano de controle baseado nas operações mais comuns de pontos de acesso de rede sem fio, permitindo que seja possível lidar com questões como:

- permitir coordenação entre pontos de acesso sem fio;
- permitir um melhor controle do *handoff* dos clientes facilitando o processo de mobilidade;
- melhorar a qualidade do enlace;
- implementar controle adaptativo para redes sem fio;
- adicionar mecanismos de configuração do plano de dados, tais como filas, canal, potência, QoS, fluxos, entre outros;
- permitir o uso de algoritmos de controle adaptados às necessidades de cada operador; e
- reduzir os custos de capital, ao permitir que o controlador gerencie redes com equipamentos de diversos fabricantes e que possa utilizar equipamentos de baixo custo.

Para tanto, definimos uma arquitetura capaz de realizar as operações necessárias e construímos um protótipo visando demonstrar a viabilidade da extensão dos conceitos de redes programáveis por software ao ambiente sem fio.

1.3 Resultados atingidos

Nesta dissertação apresentamos uma arquitetura SDN para redes locais sem fio e suas operações básicas. Descrevemos os desafios para implementação de soluções em redes sem fio e como eles podem ser superados utilizando a arquitetura proposta. Implementamos um protótipo parcial da arquitetura SDWN apresentada no capítulo 4 e o testamos em quatro estudos de caso, descritos em detalhes no capítulo 5, visando avaliar a viabilidade da solução. Um resultado parcial desta dissertação foi publicado em um artigo no *14th IFIP/IEEE Symposium on Integrated Network and Service Management* 2015 (IM 2015). Este artigo apresentou uma versão preliminar da arquitetura proposta (Moura et al. [2015]).

Avaliamos o protótipo realizando:

- o controle do processo de associação de estações sem fio aos pontos de acesso;
- o controle de fluxo de dados, objetivando garantir uma qualidade de serviço para o usuário;
- 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; e
- a identificação de interfaces sem fio que não estão transmitindo, utilizando recursos de varredura do protocolo 802.11.

Estes resultados 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 e tráfego, que demandassem respostas rápidas do controlador de forma que a latência gerada por este processo de decisão pudesse ser avaliada. Identificamos ainda que diversos protocolos auxiliares ao 802.11, como 802.11k, 802.11r e 802.11v, não estão implementados no ambiente Linux utilizado em nossos experimentos, reduzindo assim o escopo da arquitetura proposta que pôde ser implementado. Estas funções poderão ser implementadas em trabalhos futuros, bem como poderão ser testados os cenários mencionados.

1.4 Organização da dissertação

Esta dissertação apresenta cinco capítulos além da Introdução. No capítulo 2 apresentamos os conceitos básicos de redes definidas por software envolvidos no nosso traba-

lho, caracterizando sua arquitetura, seus usos e suas limitações. Destacamos ainda o protocolo OpenFlow, considerado o principal protocolo para implementação de redes SDN. No capítulo 3 apresentamos o gerenciamento tradicional (como em controladoras Wi-Fi comerciais), identificamos o conceito das redes sem fio definidas por software, apresentamos trabalhos relacionados e mostramos alguns desafios enfrentados pelos pesquisadores nesta área. Apresentamos no capítulo 4 a arquitetura proposta neste trabalho, o *Ethanol*, bem como detalhamos as operações e a implementação desta arquitetura. No capítulo 5 apresentamos os resultados de quatro estudos de caso para o protótipo construído na arquitetura do *Ethanol*. As conclusões desta dissertação são relacionadas no capítulo 6. Apresentamos ainda neste capítulo algumas sugestões para trabalhos futuros derivados da nossa arquitetura.

Capítulo 2

Redes Definidas por Software

As redes legadas tornaram-se difíceis de automatizar em razão da utilização de equipamentos com arquiteturas fechadas e inflexíveis (Anderson et al. [2005]). A incapacidade de se adaptar às novas pressões e exigências levou a um número crescente de soluções "ad hoc". A estrutura rígida das redes legadas proíbe que se possa atender às necessidades variadas dos clientes via programação de implementações customizadas, forçando que os fornecedores implementem sistemas complexos de gestão de rede para seus dispositivos. Porém, estas são soluções frágeis, pois não escalam, não evoluem na velocidade demandada pelos clientes e não interoperam com produtos de outros fabricantes (Anderson et al. [2005]).

A existência de uma cintura estreita do IP (*Internet Protocol*) significa que a arquitetura da pilha de protocolos é difícil de modificar e que novas funções têm de ser implementadas através de remendos (*patches*) na arquitetura existente (Pan et al. [2011]). Segundo os autores, tornou-se extremamente difícil suportar "as demandas cada vez maiores de segurança, desempenho, confiabilidade, distribuição de conteúdo social, a mobilidade e assim por diante" através de mudanças incrementais na pilha de protocolos.

As Redes Definidas por Software (Betts et al. [2014]) oferecem soluções capazes de reduzir alguns dos desafios acima descritos (Jammal et al. [2014]). Elas compõem um conjunto de abordagens para redes de computadores visando permitir que administradores de rede gerenciem os serviços de rede de forma mais simples, mediante uma abstração que desacopla o plano de dados do plano de controle. Esta arquitetura foi projetada para permitir que redes sejam controladas de forma mais fácil e eficiente e que os recursos de rede sejam gerenciados, utilizando algoritmos de controle sob medida para as demandas dos usuários sobre a rede (Shin et al. [2012]).

Neste capítulo descreveremos o conceito de SDN, seus usos e sua arquitetura. A

seção 2.1 trata da arquitetura de redes definidas por software. Apresentamos, a seguir, os usos para as redes definidas por software, mostrando algumas classes de aplicação de SDN. Na seção 2.2 apresentamos o protocolo OpenFlow. Finalizamos este capítulo mostrando as limitações das abordagens SDN na seção 2.3.

2.1 A abordagem de Redes Definidas por Software

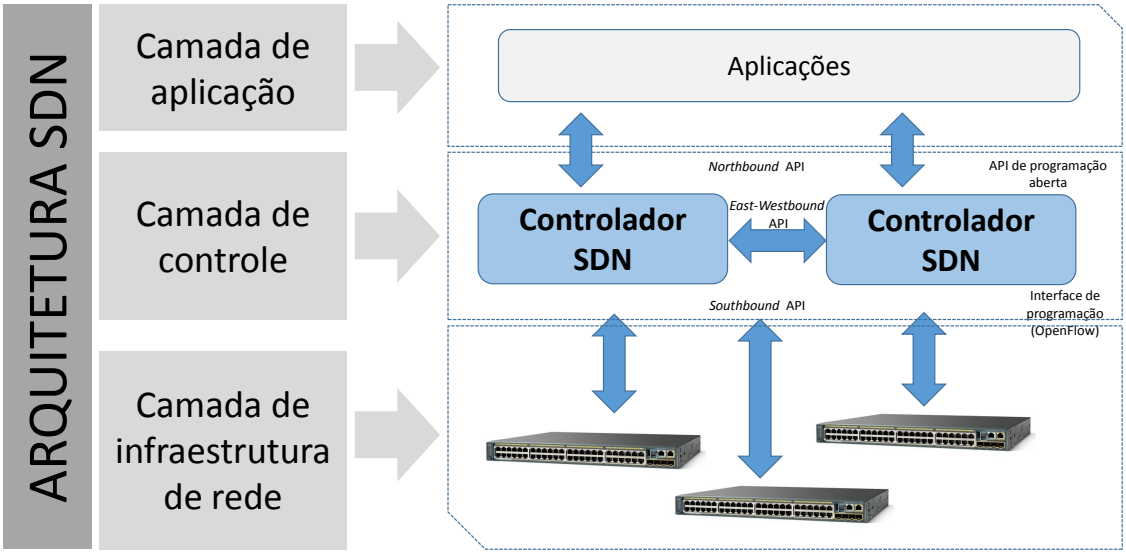
A abordagem SDN fornece ao administrador/programador de rede a capacidade de implementar regras sobre o dispositivo de rede sob a forma de uma política global e centralizada. Diversas pesquisas foram realizadas na área das redes Ethernet, como podemos ver em diversos artigos, por exemplo em Duan et al. [2012], Shin et al. [2012], Nunes et al. [2014], Xia et al. [2015], Hu et al. [2014] e Kreutz et al. [2014]. As redes SDN são caracterizadas por uma separação bem clara das funções da rede, permitindo assim a sua reprogramação via interfaces padronizadas. Sezer et al. [2013] citam que as abordagens de redes definidas por software devem apresentar quatro características: (1) a separação do plano de controle e do plano de dados, (2) um controlador centralizado com uma visão de toda a rede, (3) a presença de interfaces[1] entre os dispositivos do plano de controle (controladores) e os dispositivos do plano de dados (comutadores e roteadores) e (4) a programação da rede por aplicações externas. Kreutz et al. [2014] acrescentam que as decisões de encaminhamento nesta abordagem são baseadas em fluxos. As aplicações que consomem serviços de SDN são abstraídas das tecnologias de rede subjacentes e os dispositivos de rede são abstraídos da camada ou plano de controle de SDN. Estas abstrações garantem portabilidade e a possibilidade de implementação de novas aplicações no futuro. Por conseguinte fornecem garantias aos investimentos em serviços de rede.

Utilizando SDN, um pesquisador ou um administrador de rede ou um provedor de serviços pode introduzir um novo recurso, escrevendo um programa de software que simplesmente manipula o mapa lógico da rede (ou de uma parte desta rede). Desta forma, podemos identificar que o paradigma das redes definidas por software apresenta um grande potencial de utilização para diversas classes de utilização, como centros de dados (ou "*data centers*") (Heller et al. [2010]; Mann et al. [2012]; Shirayanagi et al. [2012]), operadoras de comunicação móvel (Gudipati et al. [2013]; Bansal et al. [2012]) e provedores de serviços (Azodolmolky et al. [2013]; Jain et al. [2013]). Pode, também, ser utilizado nos ambientes empresariais (Casado et al. [2009]; Jain et al. [2013]; Levin

[1] Os autores especificam "interfaces abertas", contudo isto faria com que não considerássemos as soluções proprietárias, como Onix (Koponen et al. [2010]) ou B4 (Jain et al. [2013]), como arquiteturas SDN.

et al. [2013, 2014]) ou residenciais (Poole et al. [2008]; Calvert et al. [2010]; Kim et al. [2011]; DiCioccio et al. [2011]; Martin & Feamster [2012]). Pode, ainda, ser utilizado para virtualização da rede de computadores (Sherwood et al. [2009]; Azodolmolky et al. [2013]; Chen et al. [2014]; Berman et al. [2014]; Al-Shabibi et al. [2014b]).

Podemos esquematizar a arquitetura de Redes Definidas por Software por três camadas ou planos distintos (Sezer et al. [2013], Foundation [2013]), mostradas na figura 2.1. Estas camadas podem ser acessadas por APIs abertas: (a) uma *camada de aplicação*, que consiste nas aplicações de rede que utilizam SDN; (b) uma *camada de controle*, que fornece controle consolidado da rede e abstrações para a camada de aplicação; e (c) uma *camada de infraestrutura*, formada pelos equipamentos de rede que encaminham os pacotes. Nesta dissertação tratamos os termos API e interface como sinônimos, uma vez que o primeiro é um subconjunto do segundo. A seguir detalhamos melhor estas camadas e interfaces:



Adaptado de [Gude et al., 2008, p.2], [Kreutz et al., 2014, p.2,4], [Betts et al., 2014, p.13].

Figura 2.1: Arquitetura das Redes Definidas por Software

- O **plano de controle** apresenta uma visão abstrata da infraestrutura completa da rede, permitindo ao administrador aplicar protocolos/políticas personalizados em todo o hardware de rede. O controlador fornece uma abstração que muitas vezes é comparada a um "sistema operacional" da rede (Sezer et al. [2013]). Existem diversos exemplos, como o NOX, POX, Beacon, Maestro, Floodlight, ONIX, entre outros (Gude et al. [2008], Erickson [2013], Cai et al. [2013], Koponen et al. [2010], Shah et al. [2013]). Haleplidis et al. [2015] esclarecem que estes contro-

ladores fornecem interfaces *northbound* para as aplicações, como esquematizado na figura 2.1.

- As interfaces ***northbound*** representam o software que provê interface entre os módulos da plataforma do controlador e as aplicações SDN em execução sobre a plataforma de rede. Estas APIs expõem abstrações do modelo de dados de rede e funcionalidades para uso pelos aplicativos de rede. Normalmente são APIs de código aberto, exceto em alguns casos, onde a solução é proprietária, como no Onix (Koponen et al. [2010]).

- O **plano de dados**, também chamado de plano de encaminhamento (*forwarding plane*), representa o hardware de encaminhamento na arquitetura de rede SDN, isto é, ele é a parte da rede que lida com o tráfego dos usuários[2]. O tráfego de dados passa através do hardware de encaminhamento, diferentemente de um hospedeiro onde o tráfego é dirigido para ele ou dele sai.

- Como o controlador precisa se comunicar com a infraestrutura de rede, certos protocolos são necessários para controlar e gerenciar estes dispositivos de rede. Esta interface é denominada ***southbound***. As tabelas de encaminhamento dos comutadores são, assim, controladas remotamente pelo controlador, como mostrado por Casado et al. [2010].

- No caso de uma arquitetura baseada em múltiplos controladores, existe ainda um protocolo de interface denominado "Leste-Oeste" ou interface ***east-westbound*** (Yin et al. [2012]), que gerencia as interações entre estes vários controladores como indicado na figura 2.1.

- A **camada de aplicações** consome os serviços fornecidos pela camada de controle. Esta camada é composta por uma ou mais aplicações que utilizam as funções oferecidas pela camada de controle para implementar o controle e a lógica de funcionamento da rede. Esta camada inclui aplicações tais como roteamento, *firewalls*, balanceadores de carga, monitoramento e assim por diante. Uma aplicação define as políticas, que são, em última instância traduzidos para instruções específicas que programam o comportamento dos dispositivos de encaminhamento.

[2]Nesta dissertação estamos generalizando o hardware de encaminhamento Ethernet pelo termo comutador, apesar dele receber na literatura diversos nomes em função de seu uso, como comutadores, roteadores, pontes etc. Quando o hardware de encaminhamento compreende um ponto de acesso de rede sem fio com vários portos Ethernet, chamamos este hardware indistintamente de roteador ou ponto de acesso.

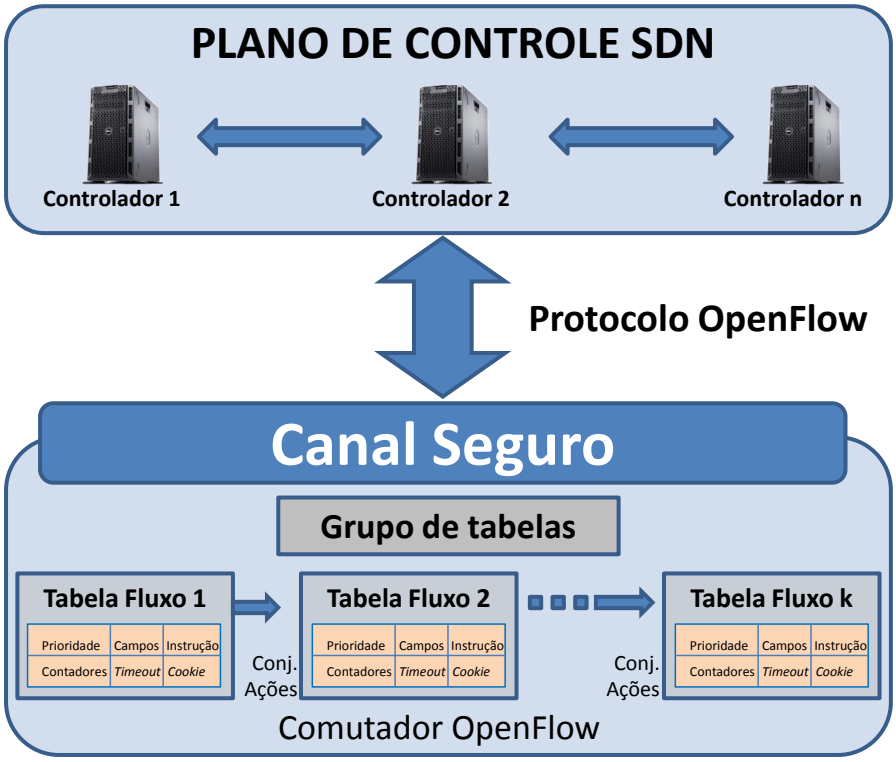
Nas descrições das arquiteturas SDN observamos que um elemento de rede é gerenciado por um controlador. Dessa forma, estas arquiteturas parecem implicar em uma centralização para as aplicações. Contudo, as redes definidas por software podem ter tanto um plano de controle centralizado quanto distribuído. Alguns exemplos de controladores distribuídos são HyperFlow (Tootoonchian & Ganjali [2010]) e ONIX (Koponen et al. [2010]).

2.2 O Protocolo OpenFlow

Uma das propostas para implementação de comunicação entre o controlador SDN e seus comutadores é o protocolo OpenFlow, que fornece uma interface *southbound* para a arquitetura SDN. O OpenFlow é um protocolo orientado a fluxo e possui abstrações para os comutadores e seus portos, usadas para controlar o fluxo de dados. Apesar do OpenFlow não ser a única implementação de SDN, pode ser considerado sua forma canônica (Casado et al. [2012]) e será utilizado na dissertação.

O OpenFlow é um protocolo aberto, inicialmente apresentado à comunidade científica pela Universidade de Stanford, que permite programar os fluxos em diferentes comutadores e roteadores (McKeown et al. [2008]), ao aproveitar os recursos existentes nos equipamentos de rede atuais. A especificação do protocolo OpenFlow é controlada por uma organização sem fins lucrativos denominada ONF (*Open Networking Foundation*), que é dirigida por um conselho de administração composto por sete empresas (Deutsche Telekom, Facebook, Google, Microsoft, Verizon, Yahoo e NTT). A maioria dos fabricantes de hardware de rede como Aruba, HP, Dell, Extreme Networks e Cisco Systems oferecem comutadores e roteadores que usam o protocolo OpenFlow (Aruba Networks [2015]; Cisco Systems [2015a]; Dell [2015]; Extreme Networks [2015]; HP Networks [2015]).

No protocolo OpenFlow, a programabilidade é possível pela atualização da tabela de fluxos dos comutadores de rede (Guedes et al. [2012]). Alterando a tabela de fluxos, um controlador OpenFlow pode mudar o comportamento dos fluxos de pacotes que chegam ao comutador. Este equipamento possui três características principais: (1) uma *tabela de fluxo* (ou mais de uma), onde uma ação pode ser associada a cada entrada de fluxo, determinando como processar cada fluxo; (2) um *canal seguro*, que liga o equipamento ao controlador remoto, permitindo que os comandos e pacotes de rede sejam comunicados de forma segura; e (3) o *protocolo OpenFlow*, que fornece uma forma aberta e padronizada para as comunicações entre controlador e comutador OpenFlow. Na figura 2.2 podemos ver estas características esquematizadas.

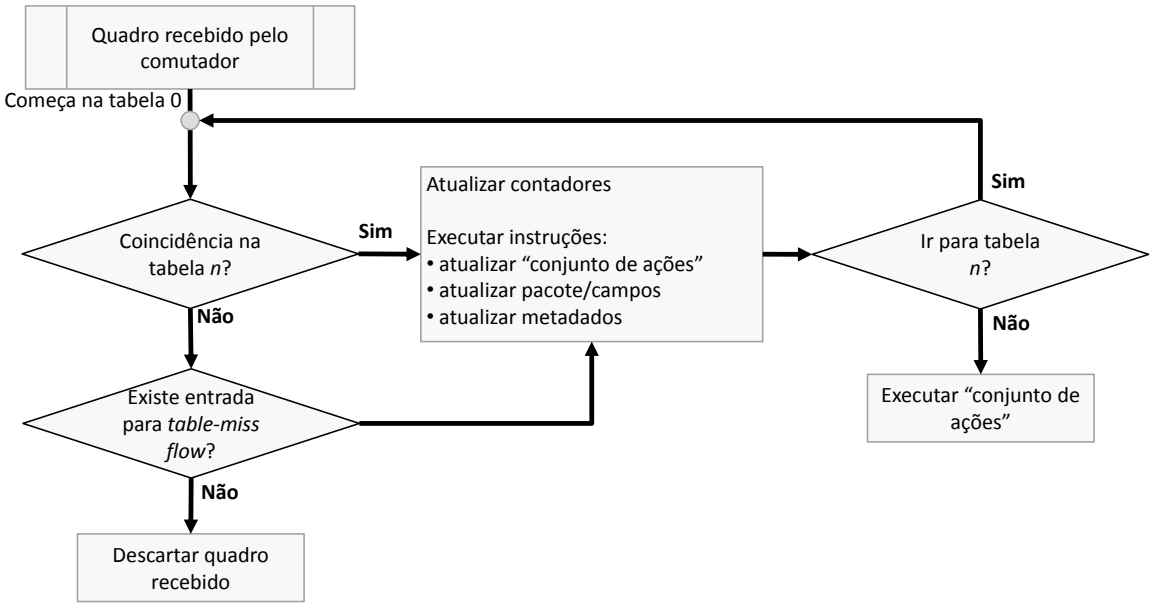


Adaptado de [Jammal et al., 2014, p.78].

Figura 2.2: Arquitetura básica do OpenFlow

Comutadores compatíveis com OpenFlow podem ser separados em dois tipos principais: OpenFlow-exclusivo e OpenFlow-híbrido (Foundation [2013]). Os comutadores OpenFlow-exclusivo fornecem apenas operações definidas no protocolo OpenFlow, ou seja, todo o fluxo de pacotes é processado pelo controlador OpenFlow. Comutadores híbridos suportam o protocolo OpenFlow (em algum nível) mas também realizam operações normais de comutação em camada 2 (enlace) e/ou camada 3 (rede).

Um comutador OpenFlow possui uma sequência de tabelas de encaminhamento, como mostrado na figura 2.2, que definem como os quadros são tratados ([Foundation, 2013, p.13-26]). Cabe ao controlador definir como estas tabelas são preenchidas e assim definir como os fluxos se comportam na rede. O controlador cria entradas na tabela, que são aplicadas caso os cabeçalhos do pacote casem com um seguinte padrão. No comutador há um processo de verificação que analisa cada tabela buscando uma regra para o quadro recebido. Três situações podem ocorrer a partir deste processo: (a) o fluxo é casado a uma regra e esta é aplicada; (2) pode haver uma regra padrão quando não existe uma coincidência nas tabelas de encaminhamento ou (3) o quadro pode ser descartado. A regra padrão normalmente consiste em enviar o quadro para o controlador, gerando o evento *PacketIn* descrito a seguir. Este processo é mostrado na



Adaptado de [Foundation, 2013, p.16].

Figura 2.3: Fluxograma detalhando o fluxo de pacotes em um comutador OpenFlow a partir da versão 1.3.0

figura 2.3.

Cada entrada na tabela de fluxos possui uma ou mais ações associadas. Estas ações (Guedes et al. [2012]) permitem *encaminhar* um pacote para um porto (ou portos) específico(s) do comutador, *alterar* uma parte do cabeçalho do pacote antes de encaminhá-lo, *descartá-lo*, *encaminhar para o controlador* o pacote para inspeção ou permitir que o controlador *crie novos pacotes* que são encaminhados à rede.

2.2.1 Controladores

Os controladores são responsáveis por programar as tabelas de fluxos dos comutadores. Existem diversos tipos de controladores SDN, como o Beacon, Floodlight, HyperFlow, Maestro, NOX, ONIX, OpenDayLight, POX, Ryu, Trema, entre outros (Gude et al. [2008], Erickson [2013], Cai et al. [2013], Koponen et al. [2010], Shah et al. [2013], Tootoonchian & Ganjali [2010], Kreutz et al. [2014]).

O POX é uma plataforma de desenvolvimento de aplicações de controle de redes SDN que roda em Python (versão 2.7 ou superior). Esta plataforma foi desenvolvida com base na especificação 1.0 do OpenFlow, com algumas adaptações para a versão 1.1. Esta é uma plataforma muito utilizada na criação de protótipos.

Com o POX é possível definir um conjunto de funções, onde cada uma responde ao evento que foi registrado. Estes tratadores de eventos executam de forma não

preemptiva. O POX oferece um amplo conjunto de eventos como:

- *ConnectionUp*, *ConnectionDown*: ocorrem, respectivamente, quando um comutador OpenFlow conecta-se ou desconecta-se do controlador;
- *PacketIn*: evento acionado quando o comutador recebe um quadro desconhecido, ou seja, este quadro que não casa com nenhuma regra instalada. Este fluxo é, então, encaminhado para o controlador;
- *FlowRemoved*: evento chamado quando um fluxo já programado no comutador é removido;
- *RawStatsReply*, *QueueStatsReceived*, *AggregateFlowStatsReceived*, *TableStatsReceived*, *PortStatsReceived*, *FlowStatsReceived*: são a resposta do comutador a uma demanda de informações sobre os fluxos de dados feita pelo controlador;
- *FeaturesReceived*, *PortStatus*, *SwitchDescReceived*: são respostas do comutador a uma demanda de informações sobre o estado ou características do comutador feita pelo controlador.

2.3 Limitações de SDN

Apesar de ser vantajoso separar fisicamente os planos de controle e dados, como vimos anteriormente, esta arquitetura traz limitações inerentes. Por exemplo, a separação do plano de dados do plano de controle, em geral, cria uma penalidade de 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[3]. A *programação* implica na capacidade de mudar ou aceitar um novo

[3]Quando uma regra já está programada no dispositivo não existe acréscimo de latência.

conjunto de funções e algoritmos, a fim de alterar o comportamento funcional do dispositivo. Já a *flexibilidade* é a capacidade de adaptação dos sistemas para suportar novos recursos não previstos. A abordagem SDN acrescenta flexibilidade ao contexto do encaminhamento de dados ao mesmo tempo que suas APIs abertas facilitam o processo de adaptação. Contudo, muitas vezes os programas dos controladores são executados em servidores de uso geral e usam linguagem de alto nível, como o Python. Mesmo utilizando controladores com executáveis compilados, como no NOX, uma rede SDN tem dificuldade em lidar reativamente com os fluxos de pacotes em enlaces de altíssima velocidade, ou seja, redes de 100Gbps ou mais (Sezer et al. [2013]). Portanto, novas abordagens são necessárias para tratar estes elementos em redes de alta velocidade. Uma forma de diminuir a latência é, sempre que possível, computar antecipadamente as regras de encaminhamento, fazendo uma instalação proativa destas regras, ao invés da forma reativa, quando a regra é criada na chegada do primeiro pacote do fluxo.

Implementações de SDN em hardware estão sendo testadas com base em requisitos cada vez mais estritos de desempenho. Nos operadores de telefonia, as restrições para recuperação de uma falha de um nó ou enlace são da ordem de 50 milissegundos em função do tráfego de voz ([Farrel & Bryskin, 2005, p.94]). Esta é uma limitação importante, pois restringe o tempo para obter uma resposta do controlador central. Diversas iniciativas utilizando processadores de uso geral, NPU/NFP (*Network flow processors* - processadores de fluxo de rede), PLDs (*Programmable logic devices*), FPGAs (*Field Programmable Gate Arrays*) e ASICs (*Application-specific integrated circuits*) estão sendo testadas para melhorar o desempenho em ambientes com SDN (Sezer et al. [2013]). Indícios apontam para a necessidade de adoção de uma solução híbrida como indicado por diversos autores (Ozdag [2012]; Bosshart et al. [2013]; Doo et al. [2012]; Narayanan et al. [2012]; Bosshart et al. [2014]).

Mesmo quando não existe esta limitação do tráfego de voz, ainda existe o problema dos dispositivos de rede perderem a conectividade com o controlador central, se o enlace primário com ele falhar. Isto implica na necessidade de se manter parte da inteligência nos dispositivos e/ou instalar caminhos alternativos pré-computados para os nós da rede. Esta abordagem pode permitir melhorar o desempenho e permitir que pelo menos parte das funcionalidades continuem funcionando independente da conexão com o controlador, porém isto gera maior complexidade de programação.

2.4 Conclusões

Como vimos, o protocolo OpenFlow é um protocolo capaz de permitir o controle do plano de dados dos equipamentos de comutação em uma rede cabeada. Diversas aplicações estão baseadas neste protocolo, contudo ainda existem desafios para esta arquitetura quando analisamos requisitos de desempenho, segurança e interoperabilidade.

As abordagens SDN típicas, como o OpenFlow, não apresentam mecanismos capazes de lidar com as complexidades do ambiente de rede sem fio. Novas maneiras de se pensar as abordagens SDN são necessárias para tratar as complexidades das redes sem fio, como apresentaremos no capítulo 3, a seguir.

Capítulo 3

Gerenciamento de redes Wi-Fi

As redes sem fio são particularmente desafiadoras e apresentam problemas adicionais aos encontrados nas redes cabeadas. Devido à natureza não confiável do meio de transmissão, devemos lidar com um maior dinamismo da rede e, em geral, maior dinamismo significa maior tráfego de controle. Quando diferentes redes sem fio não são devidamente coordenadas, elas podem interferir com os sinais da outra, degradando a experiência de uso da rede para os usuários (Ali-Ahmad et al. [2013]). Ao mesmo tempo que as redes sem fio se tornam onipresentes, principalmente em função da facilidade de sua instalação física, sofrem uma maior demanda de transmissão de dados por seus usuários que desejam altas velocidades para consumir conteúdos da web e aplicações empresariais. Em função disto, as redes sem fio estão tendendo a células menores (femto ou picocélulas) para conseguir obter maiores taxas de transferências de dados. Contudo redes mais densas tendem a aumentar a interferência (Kolias et al. [2012]; Perahia et al. [2010]).

Diversos problemas são encontrados atualmente no gerenciamento das redes sem fio. O usuário típico de uma rede local sem fio é um não-especialista, de modo que a presença maciça destas redes aumenta as chances de interferência de sinal em função de dificuldades de configuração, bem como o aumento das falhas de segurança. Mesmo se desconsiderarmos a interferência com o sinal de outros meios de transmissão, este impulso constante para se obter mais vazão provoca, muitas vezes, a diminuição do alcance da transmissão do sinal (Dely [2012]), aumentando o número de transmissores e, consequentemente, dificultando o gerenciamento e configuração dos equipamentos. Outra tendência é utilizar simultaneamente um número cada vez maior de tecnologias sem fio. Um desafio importante é a necessidade de repassar serviços utilizando diversas tecnologias sem fio, como IEEE 802.11, 3G/4G e bluetooth (Taniuchi et al. [2009]). As operadoras de telefonia móvel precisam aplicar políticas cada vez mais complexas para

garantir o acesso aos serviços e para controlar os *handoffs*[1] entre os diversos tipos de acesso. *Handoffs*, desempenho e segurança são também problemas enfrentados pelos administradores de rede em ambientes empresariais.

Muita pesquisa está sendo realizada na academia sobre a utilização de SDN em redes sem fio procurando solucionar os problemas apresentados acima. Neste capítulo começamos tratando do gerenciamento encontrado nas redes Wi-Fi atuais, inclusive descrevendo soluções comerciais (seção 3.1). Na seção seguinte (3.2) apresentamos desafios para redes sem fio definidas por software. Soluções de gerenciamento utilizando a abordagem de redes definidas por software para redes sem fio, relacionadas com o nosso trabalho, são apresentadas na seção 3.3. Concluímos este capítulo na seção 3.4 com uma comparação entre os modelos apresentados na literatura.

3.1 Gerenciamento Tradicional

O gerenciamento tradicional de redes sem fio envolve o uso de protocolos de controle e gerenciamento, bem como o uso de controladoras centralizadas produzidas pela indústria. Para prover aos usuários o serviço de compartilhamento de recursos com alta disponibilidade a um custo adequado, uma rede sem fio precisa ser propriamente configurada e gerenciada. As redes sem fio atuais são compostas por pontos de acesso isolados ou conectados a uma controladora central. Diversos protocolos, como a família IEEE 802.11, que descreveremos nesta seção, são implementados para garantir que a rede sem fio Wi-Fi funcione. Apresentamos, na seção 3.1.1, as controladoras de rede Wi-Fi e, na seção seguinte (3.1.2), os padrões IEEE 802.11 relevantes para nosso trabalho.

3.1.1 Controladoras proprietárias para redes Wi-Fi

As redes Wi-Fi dos *campi* e das empresas estão se tornando rapidamente mais centralizadas. Uma controladora central, geralmente um hardware proprietário, gere muitos pontos de acesso sem fio, alocando canais não sobrepostos, estabelecendo níveis de potência para reduzir a interferência e provendo mobilidade e autenticação de usuários em uma maneira única e consistente. Os usuários móveis podem se deslocar livremente através de um campus sem alterar o endereço IP e sem que suas conexões caiam.

[1] *Handoff* ou *Handover*, sendo o segundo termo mais usado na Europa, é o procedimento empregado em redes sem fio para tratar a transição de uma unidade móvel de uma célula para outra de forma transparente ao usuário.

As controladoras sem fio comerciais são soluções proprietárias capazes de controlar uma linha de pontos de acesso (normalmente do mesmo fabricante da controladora), que se apresentam como uma solução de gerenciamento de uma rede sem fio empresarial, simplificando a implementação desta rede mediante processos de configuração e gerenciamento centralizados. Diversos fabricantes oferecem controladoras de redes sem fio em sua linha de produtos, como Aruba Networks, Cisco Systems, Cisco Meraki, Dell, Extreme Networks e Hewlett-Packard Network (Aruba Networks [2015]; Cisco Systems [2015a]; Meraki Networks [2015]; Dell [2015]; Extreme Networks [2015]; HP Networks [2015]).

As soluções dos maiores fabricantes do mercado apresentam características similares, oferecendo: a) flexibilidade para configurar as políticas da rede sem fio, as configurações de segurança e as características dos dispositivos mediante gerenciamento centralizado; b) configuração centralizada dos pontos de acesso para controle de versão de software (os pontos de acesso possuem *firmwares* proprietários e especializados capazes de se comunicar com a controladora), detecção de interferência, alocação de canais, gerenciamento de potência de sinal transmitido; c) um sistema de firewall permitindo regras até a camada de transporte; d) um sistema de prevenção de intrusão sem fio - WIPS (*Wireless Intrusion Detection System*); e e) gerenciamento da qualidade de serviço para voz e vídeo. A tabela 3.3, no final deste capítulo, apresenta os principais pontos de comparação entre as controladoras comerciais.

Estas controladoras comerciais podem ser configuradas para alta disponibilidade utilizando outra controladora igual em modo de espera, que recebe as informações de configuração e políticas da controladora primária. Em caso de falha, esta segunda controladora passa a executar as tarefas da controladora primária (Aruba Networks [2013]; Cisco Systems [2015b]; Extreme Networks [2014]).

O uso de equipamento exclusivo e modelos de negócio obsoletos dificultam a evolução das redes móveis e das redes sem fio (Yap et al. [2010]). Os dispositivos comerciais nestas redes não são abertos e sofrem diversos problemas de escalabilidade e de flexibilidade. Para aumentar a escala, provendo a ampliação do parque, não basta comprar novos equipamentos. Estes devem ser do mesmo fabricante. Quando novos recursos são necessários nas redes, como por exemplo, novos algoritmos de roteamento, novas políticas de autenticação, etc, os clientes têm que recorrer aos fabricantes para poderem atualizar as controladoras centrais e o *firmware* dos pontos de acesso. Somente os fabricantes podem, mediante fornecimento de *patches* ou *releases*, atualizar o software destes equipamentos. Esta tendência pode ser vista ao se analisar os produtos de empresas como as acima citadas, como por exemplo em `http://www.cisco.com/c/en/us/support/wireless/`

`wireless-lan-controller-software/products-release-notes-list.html`. Os mecanismos de controle existentes nestes dispositivos não permitem customização pelos administradores de rede, que podem somente realizar configurações padronizadas e com algoritmos pré definidos pelo fabricante dos equipamentos.

3.1.2 Protocolos de gerenciamento e controle de redes Wi-Fi

Os equipamentos de rede sem fio do mercado seguem um conjunto de padrões de comunicação para que uma estação de rede sem fio não precise saber qual a marca e o modelo do ponto de acesso ao qual está se conectando. Nesta seção apresentaremos o protocolo CAPWAP e a família de protocolos IEEE 802.11.

O IEEE (*Institute of Electrical and Electronics Engineers*) publica muitos padrões importantes para o funcionamento de redes sem fio. O IEEE é uma organização sem fins lucrativos que edita publicações técnicas, organiza conferências e publica padrões baseados em consenso. Dentre estes padrões está a família de padrões denominada IEEE 802.11, que trata de redes locais sem fio de alta velocidade. Apresentamos um quadro resumo desta família na tabela 3.1. Existem, contudo, diversos outros padrões que adicionam mecanismos de gerenciamento, segurança, qualidade de serviço e testes que afetam o funcionamento de redes sem fio, como as redes Bluetooth ou as redes WiMAX. Não aprofundaremos nestes outros padrões na dissertação.

O escopo do padrão IEEE 802.11 é definir a camada de controle de acesso ao meio (MAC), prover várias especificações para a camada física (PHY) e para conectividade sem fio de estações fixas, portáteis e estações móveis dentro de uma área local (IEEE802.11 [2012]). Os padrões IEEE 802.11, 802.11a, 802.11b, 802.11g, 802.11h, 802.11j, 802.11n e 802.11z definem o funcionamento de redes locais sem fio em 2,4GHz ou 5GHz (Hiertz et al. [2010]). Os padrões IEEE 802.11ac e 802.11ad definem melhoramentos no padrão IEEE 802.11 para transmissões de alta velocidade em faixas, respectivamente, abaixo de 6GHz e 60GHz. Estes padrões definem o funcionamente da rede sem fio com diversas taxas de transmissão e diversas larguras de canal (5, 10, 20, 40, 80 e 160MHz). Iremos, nos próximos parágrafos, apresentar alguns dos padrões que são relevantes para o nosso trabalho.

Em redes IEEE 802.11, o protocolo IEEE 802.11e aprimora o protocolo IEEE 802.11 MAC implementando diferenciação de serviços (Choi et al. [2003]) e acrescentando mecanismos de correção de erro para aplicações sensíveis a atrasos, como aplicações de voz e vídeo. Este protocolo foi criado para auxiliar no encaminhamento dos fluxos de áudio e vídeo em uma rede sem fio. Com IEEE 802.11e foi introduzida uma nova função de coordenação, denominada HCF (*Hybrid Coordination Function*),

Padrão de rede	**Versões**	**Comentários**	**Situação**
IEEE 802.11	1991 1997 2007 2012	Padrão para especificações de MAC de rede local sem fio e camada física	Ativa na versão 2012
802.11a	1997	Extensão de alta velocidade para camada física na faixa de 5GHz	Incorporada à 802.11-2007
802.11b	1997 2000	Extensão de alta velocidade para camada física na faixa de 2,4GHz	Incorporada à 802.11-2007
802.11c	1998	Define o funcionamento do MAC como ponte	Incorporada a 802.1D
802.11d	1999 2001	Adiciona requisitos regionais ao padrão 802.11	Incorporada à 802.11-2007
802.11e	2000 2005	Adiciona melhoramentos em MAC para suportar QoS	Incorporada à 802.11-2007
802.11f	2000 2003	Acrescenta suporte para comunicação entre os pontos de acesso (IAPP) na operação do protocolo 802.11	Retirado pelo IEEE em 2006
802.11g	2000	Extensão de alta velocidade em 2.4GHz atingindo 54Mbps com OFDM	Incorporada à 802.11-2007
802.11h	2003	Adiciona requisitos para funcionamento na Europa	Incorporada à 802.11-2007
802.11i	2001	Adiciona requisitos de segurança ao protocolo definindo WPA e WPA2	Incorporada à 802.11-2007
802.11j	2002	Define operação em 4,9 a 5GHz no Japão	Incorporada à 802.11-2007
802.11k	2002	Define compartilhamento de informações de rádio entre estações sem fio	Incorporada à 802.11-2012
802.11ma	2003	Realizou manutenção e revisão de parte do padrão preparando para o lançamento da 802.11-2007	Incorporada à 802.11-2007
802.11mb	2007	Realiza manutenção no padrão 802.11	Terminada em 2012
802.11n	2009	Define melhorias de desempenho para maior vazão - 600Mbps com MIMO em 2,4 e 5GHz	Incorporada à 802.11-2012
802.11r	2009	Define funcionamento de transição rápida entre BSS para handoff de usuários móveis	Incorporada à 802.11-2012
802.11s	2011	Define funcionamento da rede em malha - rede mesh	Incorporada à 802.11-2012
802.11p	2010	Define funcionamento em redes veiculares	Incorporada à 802.11-2012
802.11t	2004	Apresenta práticas recomendadas para realização de avaliação e testes em redes 802.11	Retirada pelo IEEE em 2006
802.11u	2011	Define interoperabilidade entre redes 802.11 e GSM	Incorporada à 802.11-2012
802.11v	2011	Define compartilhamento de informações de gerenciamento	Incorporada à 802.11-2012
802.11y	2008	Define operação em 3650-3700MHz nos Estados Unidos	Incorporada à 802.11-2012
802.11w	2009	Define melhoria de segurança na transmissão dos quadros de gerenciamento	Incorporada à 802.11-2012
802.11z	2010	Extende funcionamento de DLS- *Direct Link Setup*	Incorporada à 802.11-2012
802.11aa	2011	Define aprimoramento no MAC para transmissões de áudio e vídeo	Ativa
802.11ac	2013	Define melhoramentos para alto desempenho, superiores a 1Gbps, para transmissões em faixas abaixo de 6 GHz	Ativa
802.11ad	2012	Define melhoramentos para alto desempenho, superiores a 1Gbps, para transmissões em faixas abaixo de 60 GHz	Ativa
802.11ae	2012	Priorização de tráfego para quadros de gerenciamento	Ativa

Tabela 3.1: Família do padrão IEEE 802.11

implementando dois métodos de acesso ao canal que permitem a definição de categorias de tráfego: HCCA (HCF *Controlled Channel Access*) e EDCA (*Enhanced Distributed Channel Access*). Por exemplo, um acesso HTTP pode ser atribuído a uma classe de prioridade baixa enquanto uma aplicação de voz na rede (por exemplo, VoWLAN)

pode ser atribuída a uma classe de prioridade alta (IEEE802.11e [2005]). Este padrão foi incorporado ao padrão IEEE 802.11 em 2007.

A comunicação entre pontos de acesso sem fio entre sistemas de múltiplos fabricantes, mediante a realização de uma única associação ao ESS (*Extended Service Set*), é tratada pelo padrão IEEE 802.11f (denominado *Inter-Access Point Protocol*). Este padrão (IEEE802.11f [2003]) define um protocolo que implementa a troca segura de informações de contexto das estações entre o ponto de acesso atual e o ponto de acesso que a estação irá migrar durante o processo de *handoff*. Estas informações transmitidas via IAPP (*Inter-Access Point Protocol*) variam conforme o tipo de mensagem. Podem conter, por exemplo, o endereço de hardware da estação e dos pontos de acesso, a chave de criptografia compartilhada, o endereço IP do servidor RADIUS (*Remote Authentication Dial-In User Service*), o nome da BSSID (*Basic Service Set Identification*) que a estação estava conectada, as informações de estado, os números de sequência das mensagens e as informações de contexto, definidas por outros padrões IEEE 802.11. Mediante a troca com sucesso das mensagens de IAPP, uma estação pode se mover entre múltiplos pontos de acesso e, ainda assim, manter a sua conexão à rede. Este protocolo responde a dois processos na rede sem fio: associação e reassociação (Huang et al. [2006]). Este protocolo foi sucedido por IEEE 802.11r.

Os protocolos IEEE 802.11k, 802.11n, 802.11r, 802.11s e 802.11v foram incorporados à nova versão do protocolo IEEE 802.11 em 2012 ([IEEE802.11, 2012, p.ix]), contudo na literatura ainda são referenciados por estes padrões antigos. Optamos, também, por referenciá-los no texto, pois facilitam a identificação dos tópicos tratados.

O padrão IEEE 802.11k (denominado *Radio Resource Measurement of Wireless LANs*) fornece mecanismos para que os pontos de acesso e as estações sem fio realizem dinamicamente medições dos recursos de rádio disponíveis e os reportem. Em uma rede com o padrão IEEE 802.11k implementado, os clientes e os pontos de acesso podem enviar entre si relatórios contendo informações sobre a sua vizinhança, informações de *beacon* e medições de características dos enlaces, permitindo que cada dispositivo possa compreender melhor o ambiente da rede sem fio em que está inserido. Com o uso deste protocolo, o IEEE pretende melhorar o gerenciamento de rede e a detecção de falhas. (IEEE802.11k [2008]; IEEE802.11 [2012])

O padrão IEEE 802.11r (IEEE802.11r [2008]; IEEE802.11 [2012]) permite suporte a realização de *handoffs* rápidos e seguros de um ponto de acesso sem fio para outro, mediante utilização de transições rápidas entre BSS (*Basic Service Set*) utilizando negociações de chaves de segurança. Este protocolo foi desenvolvido para atender aos requisitos de transição rápida capazes de suportar aplicações de voz. Já o padrão IEEE 802.21 (IEEE802.21 [2009]) trata de *handovers* entre redes heterogêneas, tais

como WiMax, IEEE 802.11 e Bluetooth.

Para permitir a configuração dos dispositivos clientes enquanto conectados a uma rede sem fio, o IEEE propôs o padrão IEEE 802.11v (IEEE802.11v [2011]). Este padrão permite que dispositivos troquem informações sobre a topologia da rede, incluindo informações sobre o ambiente de RF (radiofrequência), tornando cada cliente ciente do ambiente de rede no qual está inserido. Diversos campos de informação foram acrescentados aos quadros de gerenciamento existentes pelo protocolo 802.11v ([IEEE802.11v, 2011, p.19]). Isto permite a qualquer estação solicitar informações tão diversas como o uso do canal, as informações de interferência, a lista de SSID e a verificação se o modo de hibernação para rede sem fio em malha está habilitado ou não, entre outros. Segundo o IEEE, a distribuição deste conhecimento provoca uma melhoria global da rede sem fio, pois cada estação pode gerenciar seus parâmetros de RF baseados em informações mais completas sobre sua vizinhança.

O IEEE propôs uma alteração ao padrão IEEE 802.11 publicando o padrão IEEE 802.11aa (IEEE802.11aa [2012]). Este padrão trata da falta de mecanismos eficientes para suportar transmissões de vídeo ("*streaming*") em WLANs, não considerados pelo padrão IEEE 802.11e. Introduz um conjunto de novos esquemas de acesso ao meio, denominados GATS (*Group Addressed Transmission Service*), que estendem o funcionamento do *multicast* nas redes sem fio, considerado ineficiente e pouco confiável. Contudo o IEEE 802.11aa deixa muitas lacunas sobre como os procedimentos de otimização devem ser implementados. Salvador et al. [2013] destacam que os dispositivos compatíveis com IEEE 802.11aa possuem mecanismos codificados em *firmware*, cuja funcionalidade não pode ser alterada pelo usuário ou somente pode ser modificada mediante algumas extensões mal documentadas pelos fabricantes. A utilização de uma abordagem SDN pode ser uma solução para este problema, porém não identificamos pesquisas neste sentido.

Um mecanismo para priorizar quadros de gerenciamento é definido em IEEE 802.11ae (IEEE802.11ae [2012]), que cria um protocolo para a comunicação das políticas de priorização. Este protocolo habilita uma capacidade de priorização mais sofisticada, que equilibra a necessidade de transmitir tanto tráfego de dados quanto informações de gerenciamento de rede, melhorando o desempenho das redes IEEE 802.11. Quando os recursos de *QoS* estão ativados, ou seja, quando a QMF (*Quality-of-service Management Frame*) está habilitada, alguns quadros de gerenciamento podem ser transmitidos utilizando uma categoria de acesso diferente daquela atribuída ao tráfego de voz, a fim de melhorar a qualidade deste serviço frente aos outros fluxos de tráfego. A criação de uma política QMF diferente permite melhorar um tráfego em detrimento do outro.

Além do IEEE, diversos orgãos governamentais e organizações não governamentais propõem padrões para o funcionamento das redes de computadores. O IETF (*Internet Engineering Task Force*) é uma destas organizações. Sua missão é identificar e propor soluções a problemas relacionados à utilização da Internet, além de propor padronização das tecnologias e protocolos envolvidos. O IETF apresenta recomendações denominadas RFC (*Request for Comments*).

O IETF propôs, com a RFC 5415, o CAPWAP (*Control And Provisioning of Wireless Access Points*) como um padrão de interoperabilidade que permite a um controlador de acesso gerenciar um conjunto de APs independente da camada de enlace (Calhoun et al. [2009]). O protocolo CAPWAP define que a grande maioria das mensagens de controle da rede sem fio (que normalmente compõem o tráfego de controle entre ponto de acesso e o cliente sem fio) é direcionada para o controlador para ser processada por ele. Todas as mensagens de controle, exceto aquelas relacionadas com processamento em tempo real da camada MAC - como *Beacon Frame*, *Probe Request* e *Probe Response*, são encaminhadas pelos APs para o controlador central. O encaminhamento de todos os quadros da rede sem fio para o controlador pode implicar em sobrecarga de processamento no controlador para uma rede muito grande ou no aumento da latência.

Alguns fabricantes, como Cisco e Juniper ([Cisco Systems, 2010, p.8-2], [Juniper Networks, 2014, p.226]), oferecem suporte ao protocolo CAPWAP. O protocolo CAPWAP, contudo, apresenta limitações, pois o uso deste protocolo limita as funcionalidades disponíveis nos dispositivos sem fio. O CAPWAP define uma abordagem de "mínimo denominador comum" para o controle dos pontos de acesso, ou seja, o protocolo trata somente os quadros de gerenciamento do IEEE 802.11. Desta forma, o CAPWAP não permite a implementação de todos os diferentes algoritmos de controle e todas as configurações necessárias para um dispositivo Wi-Fi. Estas operações são fornecidos pelos fabricantes como comandos e funções proprietários. A empresa Aruba argumenta que um sistema fechado e proprietário é necessário para suportar o ritmo de desenvolvimento exigido pelo mercado (Aruba Networks [2009]), citando, como exemplo, que os protocolos desenvolvidos pelo IEEE como 802.11k, 802.11r, 802.11v e 802.11y, não são levados em conta pelo CAPWAP. Esta empresa até o presente momento não fornece suporte ao CAPWAP em seus equipamentos.

Os protocolos IEEE 802.11 garantem a compatibilidade na comunicação entre os pontos de acesso e as estações sem fio. A abordagem do CAPWAP permite alguma flexibilidade em relação às mensagens de gerenciamento, contudo também não atende às necessidades dos usuários. Portanto o gerenciamento tradicional não é suficiente para resolver todas as dificuldades encontradas nas redes sem fio (Sezer et al. [2013]).

Na próxima seção apresentamos alguns dos desafios a serem enfrentados nas redes sem fio defindas por software.

3.2 Desafios em redes sem fio definidas por software

A programação de redes sem fio com APIs abertas pode aumentar a robustez e o desempenho das WLANs de grande escala, reduzir o seu custo e permitir a criação de novos serviços. Existe um benefício significativo ao estender aos APs a capacidade de programação nos moldes do SDN, permitindo otimizações, tais como a seleção de melhores canais, o ajuste de potência de transmissão em cada ponto de acesso, a mobilidade mais rápida dos clientes, a melhoria na rastreabilidade e a aplicação de políticas de segurança e *QoS*. Estes serviços são em parte fornecidos para os ambiente empresariais por controladoras de fabricantes como Cisco, Aruba, Meraki, Meru etc (Aruba Networks [2013, 2014]; Meraki [2009]; Cisco Systems [2015b]; Meru Networks [2014]). Ao observamos estas soluções comerciais, percebemos que há uma tendência para o controle central da rede sem fio. Os produtos destas empresas demonstram que esta abordagem é viável.

No entanto, essas controladoras não atendem completamente às necessidades dos administradores de rede. As controladoras comerciais são proprietárias e são projetadas para trabalhar com produtos específicos, pois gerenciam dispositivos geralmente de um único fabricante. Estes equipamentos possuem interfaces fechadas e genéricas. Por exemplo, o tempo de *beacon* é um parâmetro configurado estaticamente na maioria das controladoras comerciais. Porém existe um conflito: menor intervalo entre os *beacons*, mais rápido o descobrimento dos roteadores na rede e melhor o funcionamento em *roaming* de usuários, contudo o envio de *beacons* consome banda que poderia estar sendo utilizada para transmitir dados de usuários. Uma maior quantidade de *beacons* também reduz o tempo de hibernação das estações sem fio, que por isto consomem mais energia. A configuração dinâmica deste parâmetro recebendo valores diferenciados nas áreas de cobertura dos APs poderia permitir que cada ponto de acesso se adapte ao seu ambiente específico. Os programas das controladoras não permitem que o administrador de rede possa adequar a solução existente à uma demanda específica, mediante, por exemplo, a implementação de um novo algoritmo de controle ou alteração do algoritmo atual.

As redes sem fio possuem diversas particularidades em relação às redes cabeadas em razão das características variáveis dos enlaces e maior mobilidade dos usuários. A

mobilidade na rede sem fio, por exemplo, apresenta desafios adicionais à mobilidade encontradas nas redes cabeadas, como por exemplo na migração de máquinas virtuais (VM). A mobilidade de VM ocorre de forma controlada, dentro de um mesmo domínio administrativo e normalmente dentro da mesma subrede. Os usuários móveis em uma rede sem fio não seguem qualquer planejamento (identificável pelo administrador de rede). Eles podem mudar de subrede em função da troca de ponto de acesso e até mesmo podem trocar de tecnologia de conexão e de domínio administrativo. As abstrações e primitivas propostas pelas abordagens SDN para redes com fio (e particularmente com relação ao uso do OpenFlow) não são suficientes para realizar um controle adequado das redes sem fio.

A utilização de abordagem SDN às redes sem fio dá origem a iniciativas de Redes sem Fio Definidas por Software ou SDWN (*Software Defined Wireless Networks*). Nas SDWN o plano de controle é desacoplado do plano de dados da rede sem fio (Chaudet & Haddad [2013]). Isto permite abstrações de programação que modelam os aspectos fundamentais de uma rede sem fio, como o gerenciamento de estado, o provisionamento de recursos, o monitoramento de rede e a reconfiguração da rede (Riggio et al. [2015]).

Esta seção apresenta alguns dos desafios impostos pelas redes sem fio e como isto influencia o projeto de um plano de controle programável. Apresentamos nas subseções a seguir indicações que o controle de redes sem fio empregando uma abordagem SDN é algo desafiante, porém promissor, e que é, portanto, importante dedicar esforços na separação do plano de controle do plano de dados também na esfera das redes sem fio.

3.2.1 Características variáveis dos enlaces

Ao contrário das rede cabeadas, onde os enlaces possuem vazão fixa e taxas de erros baixas, nas redes sem fio estas características variam para cada quadro transmitido. Desta forma, um controlador SDWN deve considerar as características do enlace - tais como taxas de bits, atraso, taxa de perdas, flutuações no atraso dos pacotes ("*jitter*"), etc - para tarefas como roteamento em redes em malha, alocação de canais ou escalonamento de transmissão de clientes sem fio.

A qualidade da transmissão é afetada fortemente pelo congestionamento e pelas interferências, desta forma identificar a vizinhança local (próxima a cada cliente) é importante (IEEE802.11v [2011]). A implementação do Jigsaw (Cheng et al. [2006]) mostrou as dificuldades em se obter uma visão global de uma rede sem fio, relacionadas ao posicionamento correto de monitores de rede, à sincronização em larga escala, à unificação de quadros e à reconstrução de fluxos em múltiplas camadas.

O acréscimo da estimação da qualidade e da obtenção das características parti-

culares do enlace sem fio geram complicações adicionais ao processo de descobrimento da topologia da rede. Os protocolos IEEE 802.11k e IEEE 802.11v podem auxiliar na obtenção destas informações, entretanto estes protocolos se baseiam em uma decisão local. Uma abordagem SDN permite uma visão global da rede, melhorando a qualidade das decisões. O controlador pode receber de seus roteadores sem fio informações das estações conectadas. O controlador pode ainda solicitar às estações sem fio informação da sua vizinhança.

A comunicação entre o controlador e os elementos de rede sofre dos mesmos problemas enfrentados pelos dados dos usuários. Em redes sem fio em malha - redes "mesh" ou WMN (*Wireless Mesh Networks*), a falta de confiabilidade dos canais de rádio pode impedir temporariamente as comunicações com o controlador e provocar a falha de protocolos de roteamento. A arquitetura de rede totalmente centralizada impõe rigorosos limites sobre a latência entre o controlador e o AP, pois a decisão de controle deve chegar ao AP antes que as informações do estado do canal, a partir das quais a decisão foi tomada, tornem-se obsoletas.

3.2.2 Mobilidade

As Redes sem Fio Definidas por Software (SDWN) devem ser capazes de gerenciar a mobilidade dos clientes. Elas devem controlar como os usuários se associam aos pontos de acesso, bem como identificar quando deve ocorrer um *handoff* de um cliente de um ponto de acesso para outro. Isto é desejável para implementação de balanceamento de carga e para o controle da topologia, bem como para reduzir o tempo de transferência de uma estação de um ponto de acesso a outro. Para isto é necessária a coordenação das tarefas de gerenciamento, tais como a autorização/autenticação, a associação e a atribuição de endereços IP.

A mobilidade dos clientes é um problema significativo em redes sem fio, como mostrado pelos diversos padrões que tratam deste assunto, como os protocolos IEEE 802.11f, 802.11r e 802.21. Contudo estes padrões não estão implementados em todos os equipamentos. Mesmo que estivessem ainda não resolvem todos os problemas, pois baseiam-se em decisões locais não ótimas que muitas vezes não representam um benefício para a rede global. Por exemplo, em uma rede coberta por dois pontos de acesso, as estações conectam-se a qualquer um dois dos APs. Contudo se parte destas estações atendem aos padrões IEEE 802.11b e outra parte é 802.11n, é melhor segregar as estações, conectando um tipo a um AP e outro tipo a outro AP. Esta segregação não pode ser feita usando exclusivamente os padrões IEEE 802.11.

As decisões nos protocolos de rede sem fio são realizadas pelas estações ou pelo

ponto de acesso individualmente. Estes determinam, localmente e de forma distribuída, as ações a serem realizadas. Nem sempre estas decisões locais são favoráveis ao funcionamento global da rede. Neste contexto, a existência de um controlador com visão global e com conhecimento das políticas aplicáveis pode melhorar a qualidade das decisões tomadas. ODIN (Suresh et al. [2012]) e Mobileflow (Pentikousis et al. [2013]) exploram estes aspectos de mobilidade e *handoff* de usuários em redes sem fio com SDN. Contudo a latência de comunicação pode não ser favorável a esta centralização.

3.2.3 Qualidade de serviço

A especificação atual do OpenFlow[2] possui suporte simples aos mecanismos de qualidade de serviço de rede. A especificação do OpenFlow permite atribuir um fluxo de dados a uma fila de prioridades existente no comutador utilizando a estrutura de ação *Set-Queue*. Contudo com o OpenFlow é possível, utilizando o recurso de *meter action*, implementar algumas operações de *QoS*, como limitar a taxa de tráfego enviado ou recebido por uma interface de rede. Porém este recurso é opcional ([Foundation, 2013, p.20-22]) e, normalmente, não está disponível nos comutadores OpenFlow comerciais.

Somente estas configurações não garantem uma qualidade mínima para experiência do usuário, QoE (*Quality of experience*). As dificuldades encontradas para garantir priorização em redes sem fio ficam evidenciadas pelos diversos protocolos da família IEEE 802.11 que procuram solucionar este problema. Os protocolos IEEE 802.11e, IEEE 802.11aa e IEEE 802.11ae definem diversas alterações na camada de enlace para adequá-la aos requisitos das aplicações sensíveis a atrasos. Por exemplo, é criada uma nova função de coordenação com novos métodos de acesso ao canal. São propostas também melhorias nas comunicações *multicast* e nas transmissões das mensagens de gerenciamento, com o estabelecimento de fluxos de dados com melhor alocação dos tempos de transmissão.

Utilizando um controlador SDN, devemos ser capazes de configurar parâmetros de rede que influenciam positivamente na qualidade dos enlaces de rede com e sem fio. Garantir um *QoS* adequado em redes sem fio é mais complexo que em redes com fio, uma vez que o tempo de entrega de pacotes e a vazão podem variar em função da natureza dinâmica dos enlaces, bem como da contenção no enlace sem fio, uma vez que este é um meio de transmissão por difusão (*broadcast*). A família de protocolos IEEE 802.11 deixam a decisão para cada estação, contudo um controlador SDWN, por possuir uma visão global da rede, pode ser capaz de tomar melhores decisões sobre o comportamento da rede como um todo.

[2]Protocolo versão 1.5.1 - publicado em abril de 2015.

3.2.4 Virtualização

A virtualização de redes permite o compartilhamento de múltiplas redes lógicas em uma mesma infraestrutura física, mantendo seus fluxos isolados e permitindo a utilização de endereçamento e mecanismos de encaminhamento diferenciados entre estas redes lógicas. Sherwood et al. [2009] utilizam uma abordagem SDN, denominada FlowVisor, para obter este tipo de segmentação em redes cabeadas, dividindo um elemento de comutação em cinco dimensões: vazão, topologia, tráfego, CPU do dispositivo e tabelas de encaminhamento. Estas mesmas dimensões, com exceção das tabelas de encaminhamento, estão presentes em pontos de acesso sem fio e podem ser controladas por uma entidade centralizada.

O OpenVirteX (Al-Shabibi et al. [2014a]) estende o FlowVisor fornecendo uma infraestrutura como serviço (IaaS) em ambientes SDN. Desta forma, permite aos administradores de rede criar e gerenciar redes SDN virtuais. Esta ideia pode ser transposta para o ambiente de rede sem fio em uma SDWN, de modo que uma rede possa ser tratada como serviço, sendo instanciada, migrada e removida.

Um *framework* SDN proposto por Suresh et al. [2012], denominado ODIN, busca simplificar a implementação de serviços de alto nível em WLAN empresarial, tais como AAA (*Authentication, Authorization and Accounting*). Através da introdução de pontos de acesso virtuais, o usuário tem a impressão que está conectado ao seu ponto de acesso em toda a área de cobertura. CloudMAC (Dely2012) propõe a virtualização dos pontos de acesso, fazendo com que cada usuário possa ter um AP próprio independente do local de conexão. As soluções apresentadas funcionam bem quando a cobertura é de uma única área administrativa. Elas não funcionam se tiver que operar com duas ou mais áreas. Existem ainda limitações de hardware que impedem que cada usuário tenha parâmetros distintos na rede sem fio. Ou seja, nem todos os parâmetros da uma rede sem fio podem ser virtualizados para os usuários. Por exemplo, existe limitação da quantidade de SSID que pode ser transmitida por um ponto de acesso. Os equipamentos possuem número reduzido de rádios, portanto o canal não pode ser virtualizado.

3.2.5 Segurança

Em ambientes de rede sem fio, a segurança é um tópico importante. Qualquer cliente sem fio que estiver no alcance de uma transmissão pode interceptar a comunicação, buscando ouvir o que está sendo transmitido, ou pode tentar interferir na transmissão, por exemplo, alterando seu conteúdo ou impedindo sua transmissão.

Um ambiente SDWN fornece a capacidade ao administrador de rede de monitorar os fluxos de dados de seus dispositivos. Este monitoramento fornece uma visão da situação da rede, provendo meios para detectar intrusos e atividades anormais. O tráfego corrente pode ser comparado com dados históricos ou com padrões e estatísticas de tráfego considerados aceitáveis ou maliciosos, de modo que um controlador possa decidir se aquele tráfego atende a um padrão esperado. Por exemplo, o OpenSketch (Yu et al. [2013]) pode ser utilizado para monitorar comportamentos anômalos na rede. Ele fornece um *pipeline* de três estágios, abrangendo *hashing*, filtragem e contagem, que suporta diversas tarefas flexíveis de medida utilizando *sketches*[3]. Os resultados de sua utilização mostram que medidas complexas podem ser obtidas utilizando uma abordagem SDN e, dessa forma, podem ser aplicadas, também, na caracterização do tráfego de rede sem fio.

Uma importante tarefa de segurança em redes corporativas é a detecção de pontos de acesso não autorizados (*rogue APs*). Estes dispositivos degradam o desempenho da rede sem fio corporativa e expõem a rede a acessos não autorizados. A detecção destes pontos de acesso requer a cooperação entre os pontos de acesso da corporação. Esta tarefa pode ser implementada com a utilização de um controlador central SDWN, necessário para o processamento do perfil de tráfego e para triangulação do invasor. Uma forma de restringir um *rogue AP* é negando simultaneamente o encaminhamento dos pacotes por ele originados na rede sem fio e na rede cabeada. Este tipo de resposta mostra a interdependência entre as abordagens SDN para redes com e sem fio.

A utilização de uma abordagem SDWN também facilita a realização de autenticação e autorização entre múltiplos provedores. O controlador de cada provedor pode expor uma API comum com as informações dos usuários e parâmetros de SLA (*Service Level Agreement*) que os demais provedores devem atender.

3.2.6 Localização

A localização de uma estação sem fio (usuário) é um elemento importante para fornecer serviços baseados em localização. A localização é importante também para decisões de *handoff* e para segurança de rede. Uma estação sem fio pode fornecer ao controlador SDWN uma indicação da direção de seu movimento, permitindo que sejam realizados *handoffs* mais duradouros. Algoritmos de detecção de ataques e identificação de *rogue APs* podem identificar acessos não autorizados e permitir ao controlador realizar contra medidas. A informação de localização pode ser utilizada neste caso, por exemplo, para

[3] *Sketches* são estruturas de dados que permitem a sumarização eficiente de grandes fluxos de dados. (Cormode & Muthukrishnan [2005])

identificar a posição dos intrusos, o que é essencial para realização de contra medidas físicas, como o envio de pessoal de segurança ao local. A localização é importante também para tomada de decisões de roteamento geográfico, como por exemplo em uma grande rede de sensores em malha onde é necessário rotear os pacotes baseado na posição dos nós.

Chintalapudi et al. [2010] mostram que a localização em ambientes internos (*indoor*) é possível sem que haja necessidade de um esforço de implementação prévia. O algoritmo de localização, denominado EZ, recebe como parâmetros medidas de RSS (*Received Signal Strength*) dos pontos de acesso conhecidos, fornecidas por agentes instalados nas estações sem fio ou estas estações informam sua posição geográfica quando conhecida. Estas coordenadas podem ser obtidas por dispositivos de GPS (*Global Positioning System*), por exemplo, em posições próximas a portas ou janelas da construção. Utilizando os protocolos IEEE 802.11k e/ou IEEE 802.11v, estas leituras de RSS e GPS poderiam ser solicitadas às estações pelos pontos de acesso conectados ao controlador SDWN, permitindo assim a implementação de mecanismos de localização sem o uso de hardware ou software especiais nas estações.

3.2.7 Configuração

A utilização de redes sem fio apresenta dificuldade para sua adequada configuração e gera diversos problemas de gerenciamento. Nas empresas, a quantidade e a diversidade de equipamentos também representam problemas para os administradores de rede. Similarmente, diversos problemas são encontrados nas residências.

As redes domésticas estão se tornando cada vez mais comuns. As operadoras de banda larga/provedores de internet têm fornecido junto com seu serviço de provimento de acesso um dispositivos de acesso a rede que já suporta a rede sem fio IEEE 802.11. Em uma rede doméstica, um dos principais objetivos é permitir compartilhamento transparente de dados entre os vários dispositivos na casa, restringindo o acesso a outros e provendo uma segurança adequada para a rede interna. Um dos gargalos na concretização deste objetivo é a necessidade de configuração de rede pelo usuário final, normalmente um não especialista, que não está disposto a se dedicar a esta tarefa (Chetty et al. [2007]; Poole et al. [2008]). Feamster [2010]; Yiakoumis et al. [2011]; Kumar et al. [2013a] propõem o uso de SDN para superar estes problemas e sugerem a terceirização da configuração e gerenciamento de rede. Aplicações de *streaming*, fornecidas por empresas como YouTube ou Netflix, podem se beneficiar de uma rede personalizada, oferecendo uma melhor experiência de uso ao usuário.

O maior custo de uma operadora de telecomunicações hoje é o suporte aos usuá-

rios. Este serviço que corresponde a cerca de 35% do $OPEX$ para as operadoras móveis europeias, segundo o estudo de Capgemini Consulting [2009]. Este custo é cinco vezes maior que o segundo maior custo apontado pelo estudo. Portanto, reduzir as visitas técnicas traz benefícios para a operadora. O usuário também se beneficia ao receber um serviço que de outra forma não seria possível, como diagnóstico da rede (Aggarwal et al. [2009]; DiCioccio et al. [2011]; Sundaresan et al. [2013]), ações proativas de melhoria de segurança (Feamster [2010]) ou melhoria do desempenho (Kim et al. [2011]). Outra abordagem para aliviar a carga de configuração de rede dos usuários domésticos é virtualizar o AP, fornecendo a cada usuário seu próprio ponto de acesso virtual personalizado (Suresh et al. [2012]). O acesso a rede poderia estar disponível em todos os locais cobertos por sua provedora de acesso, como proposto em anyFi.net (`http://anyfi.net/`).

As redes empresariais também demandam um grande esforço de configuração e administração. Os produtos comerciais oferecidos por empresas como Cisco, Aruba, Meraki e Meru (Aruba Networks [2013, 2014]; Meraki [2009]; Cisco Systems [2015b]; Meru Networks [2014]) permitem realizar estas operações de forma centralizada, porém são soluções proprietárias que não interoperam. Estas soluções são caras e fechadas, não permitindo ao administrador implementar uma solução sob medida para sua necessidade. A utilização de APIs abertas, como OpenFlow, pode resolver este problema. Contudo, o OpenFlow não opera sob os parâmetros de rede sem fio, apesar de poder ser utilizado para transporte de mensagens específicas para o controlador.

Em redes sem fio em malha, um dos desafios encontrados, em contraste com as redes sem fio IEEE 802.11 empresariais e as redes celulares, é a ocorrência de mudanças na topologia. Os nós de uma rede "mesh" podem ser móveis e podem entrar e sair da rede com frequência.

3.3 Trabalhos relacionados de gerenciamento SDN

A abordagem SDN tem interessado a comunidade de rede sem fio também. De fato, um número crescente de empresas que trabalham na área de comunicações sem fio e de dispositivos móveis aderiram a iniciativas relacionadas a SDN. Por exemplo, a Verizon, a Nokia Siemens Networks, a Ericsson e a Netgear são membros da ONF. A ONF possui entre suas comunidades técnicas um Grupo de Trabalho (WG) específico para Redes sem Fio e Dispositivos Móveis (`https://www.opennetworking.org/images/stories/downloads/working-groups/charter-wireless-mobile.pdf`).

Além do grande interesse da indústria no uso de SDN para gerenciamento, encon-

tramos uma grande quantidade de artigos científicos no tema. Nas subseções a seguir, apresentamos os trabalhos atuais mais relevantes para nossa proposta.

3.3.1 Redes IEEE 802.11

As redes sem fio definidas pelo protocolo IEEE 802.11 são bastante comuns e apresentam diversos desafios de configuração e de gerenciamento. Diversos trabalhos abordam aspectos como virtualização, configuração, melhoria de sinal e controle de aplicações, utilizando a abordagem de redes definidas por software. Nesta seção apresentamos alguns destes trabalhos.

A proposta apresentada por Dely et al. [2012] realiza o processamento, em servidores dentro de centros de dados, dos quadros encaminhados aos pontos de acesso sem fio. Nesta proposta, denominada *CloudMAC*, a ligação entre os pontos de acesso virtuais, VAPs rodando no controlador, e os pontos de acesso físicos, compatíveis com IEEE 802.11, é gerenciada usando OpenFlow. O processamento dos quadros, bem como a criação de estruturas de gestão, é tratada nos APs virtuais. A separação dos planos de controle e de dados permite que novos serviços possam ser facilmente implementados em linguagens de programação de alto nível. Apesar dos testes mostrarem a viabilidade de funcionamento da solução, ainda é necessário uma avaliação acurada de desempenho para um ambiente de produção, pois o tráfego da rede sem fio pode saturar os enlaces com os centros de dados.

O *Odin* trata da mobilidade das estações em redes WLAN usando o conceito de redes virtuais. Este *framework* proposto por Suresh et al. [2012] abstrai um AP virtual que controla o estado de associação de cada cliente e separa esta condição do AP físico. Para o controlador, os clientes associados a um AP são percebidos como conexões a portos virtuais. A arquitetura consiste em um sistema mestre, construído sobre um controlador OpenFlow, e agentes instalados nos APs. Esta abstração facilita, segundo os autores, o tratamento da associação e da mobilidade dos clientes, permitindo que a infraestrutura realize seu *handoff* sem precisar acionar os mecanismos de reassociação, pois para a infraestrutura o cliente continua conectado ao mesmo porto virtual, eliminando o trabalho de autenticação e de reconfiguração quando ele se move.

O *OpenRF* fornece uma interface para a camada física, alterando o processamento de sinal MIMO (*Multiple-Input and Multiple-Output*). A proposta apresentada por Kumar et al. [2013b] permite que os pontos de acesso utilizem o mesmo canal mediante o cancelamento mútuo da interferência de um AP sobre os clientes de outro AP da rede. Ao mesmo tempo, utilizam da técnica de *beamforming* sobre o sinal ao enviá-lo aos clientes, melhorando a potência do sinal recebido e a taxa de transmissão para os

clientes. Como as alterações são realizadas nos APs, somente a qualidade de serviço do tráfego de *downlink* é melhorada. Para implementar o *OpenRF*, a separação do plano de dados e do plano de controle foi necessária.

No trabalho apresentado por Kim et al. [2011] é proposta uma solução para controle de redes sem fio domésticas, objetivando maior visibilidade sobre o uso de recursos por aplicações e usuários, ao mesmo tempo, provendo meios para limitar o uso desses recursos. Martin & Feamster [2012] apresentam o *CAPS*, uma proposta que também visa maximizar a experiência dos usuários. Segundo os autores, os usuários podem ser priorizados com base na atividade específica que estão realizando. As ideias propostas nestes dois artigos podem ser generalizadas para uma rede sem fio onde se deseja níveis distintos de acesso para usuários corporativos, para aplicações e também para usuários visitantes. Um trabalho semelhante é apresentado por Patro & Banerjee [2014], que propõem uma arquitetura SDN de gerenciamento de redes sem fio residenciais denominada CoAP (*Coordination framework for Open APs*). Esta arquitetura fornece uma API para redes IEEE 802.11, capaz de configurar parâmetros de RF dos pontos de acesso e obter informações de uso. Os pontos de acesso são controlados por um controlador central baseado na nuvem.

Riggio et al. [2015] propõem uma abstração utilizando os conceito de SDWN para redes sem fio, fornecendo um modelo de classes cobrindo diversos aspectos do funcionamento destas redes. Na arquitetura proposta pelos autores, denominada *EmPOWER*, são apresentadas primitivas de funções para gerenciamento de estado (comportamento) da rede, provisionamentoe de recursos, monitoramento de rede e configuração. Os autores criam, para prova de conceito, um protótipo para redes IEEE 802.11 controlado por uma SDK (*Software Development Kit*) em Python. Os pontos de acesso são alterados com a instalação de um agente baseado em *Click* (`http://www.read.cs.ucla.edu/click/click`), que permite abstrair a conexão de cada usuário como um ponto de acesso virtual. A arquitetura proposta por estes autores é bastante similar ao *Ethanol*. Contudo, o *Ethanol* é capaz de tratar aspectos de *QoS*, apresenta para o desenvolvedor um conjunto maior de parâmetros e é capaz de expor informações obtidas a partir das estações sem fio. Estes três aspectos não são tratados com o *EmPOWER*. Contudo a abordagem adotada para o *EmPOWER* foi mais genérica, com objetivo de permitir que fosse aplicada a qualquer tipo de rede sem fio.

3.3.2 Redes celulares

Com a evolução das redes de telefonia móvel celular, a demanda dos usuários modificou-se de uma demanda por conversação de voz para a demanda por uma diversidade de tráfego envolvendo dados, voz e vídeo, comparável ao encontrado nas redes cabeadas. Desta forma, diversos desafios surgiram em função do aumento dos requisitos dos usuários, inclusive de maiores taxas de transmissão, da necessidade de maior cobertura, da mobilidade dos usuários e da complexidade da infraestrutura necessária para suportar esta demanda. Diversas pesquisas têm mostrado a viabilidade da adoção de conceitos de redes definidas por software para estas redes. Nesta seção apresentamos algumas delas.

Gudipati et al. [2013] apresentam uma proposta para uma rede de acesso via rádio - RAN (*Radio Access Network*) capaz de modificar como a rede sem fio se comporta com *handovers*, na atribuição de grupos de blocos de recursos para cada fluxo de cliente e na atribuição de potência para cada bloco de recursos em cada estação base. A proposta consiste na abstração das estações base instaladas em uma área geográfica como uma grande estação base virtual, composta de elementos de rádio (as estações de base físicas) e os recursos de rádio como uma grade tridimensional (espaço, tempo e *slots* de frequência). Esta separação do plano de controle e do plano de dados permite, conforme argumentam os autores, a melhoria do funcionamento de redes LTE densas. A implementação do *SoftRAN* permite realizar o balanceamento de carga e otimiza a utilidade global dos fluxos de rede celulares. Estas duas operações seriam extremamente difíceis em um ambiente onde as decisões fossem tomadas localmente em cada estação base.

O *CloudIQ* realiza o processamento de sinais da rede 3GPP (*3rd Generation Partnership Project*) em um controlador centralizado (situado à distância máxima de 20km das antenas) nos moldes do SDN. Nesta implementação, Bhaumik et al. [2012] dividem as estações base em grupos, a fim de garantir que o controlador será capaz de atender aos requisitos da rede. O processamento dos sinais é escalonado para o controlador, incluindo todo o processamento digital até a camada física. Nas estações-base funcionam somente os componentes de RF e as antenas. O balanceamento de carga de processamento nos servidores do controlador permite reduzir os custos.

O núcleo das redes celulares apresenta problemas em função do uso de equipamentos caros, proprietários, inflexíveis e com protocolos de controle complexos. O *SoftCell* (Jin et al. [2013]) é uma proposta de uma arquitetura escalável que suporta as políticas para controle de dispositivos móveis no núcleo das redes celulares, utilizando comutadores e servidores tipo *commodities*, mediante a utilização do conceito das redes

definidas por software. *SoftCell* permite aos operadores da rede realizarem políticas de serviço de alto nível que direcionam o tráfego através de sequências de *middleboxes* com base em atributos dos assinantes e das aplicações, mediante aplicação de regras às tabelas de encaminhamento dos comutadores. Para minimizar o tamanho destas tabelas, *SoftCell* agrega todo o tráfego ao longo do percurso utilizando múltiplas dimensões (como, por exemplo, referenciando as políticas de serviço, a estação base e o dispositivo móvel). Como a maior parte do tráfego, segundo os autores, origina-se dos dispositivos móveis, a classificação dos pacotes é feita diretamente nos comutadores de acesso, dentro das estações de base. As regras atribuídas a estes equipamentos garantem que os pacotes que pertencem à mesma ligação trafeguem pela mesma sequência de *middleboxes* em ambas as direções do fluxo, mesmo na presença de mobilidade. Nesta arquitetura, os usuários não necessitam de alteração, somente o núcleo da rede é afetado.

A proposta de Pentikousis et al. [2013], denominada *Mobiflow*, é a transformação dos elementos de controle de usuários em redes celulares, atualmente em hardware, para serviços virtualizados em software. Os autores propõem o uso de um motor de encaminhamento e de um controlador capazes de substituir, em uma estação base, as funções atualmente realizadas por servidores dedicados para AAA. Desta forma estes equipamentos são transformados em software, capazes de serem virtualizados em qualquer local da nuvem. Com o *Mobiflow* é possível fazer uma implantação incremental, substituindo gradualmente uma planta, ao mesmo tempo suportando as funcionalidades de rede requeridas pelas operadoras, como o tunelamento em camada de rede e a tarifação flexível, mantendo um nível de desempenho compatível com os requisitos das operadoras. Uma abordagem equivalente, visando otimização de recursos em redes celulares e IEEE802.11, é feita com a arquitetura CROWD (*Connectivity management for eneRgy Optimised Wireless Dense networks*), proposto por Ali-Ahmad et al. [2013].

O *OpenRoads*, apresentado por Yap et al. [2010], é uma plataforma de rede móvel sem fio que permite a pesquisa experimental e implantação de redes e serviços em uma infraestrutura de redes sem fio. *OpenRoads* utiliza o protocolo OpenFlow para separar o controle do plano de dados. Os autores propõem o que denominam *OpenFlow Wireless*, construído no topo do OpenFlow e utilizando os recursos do FlowVisor, para permitir segmentar ou virtualizar o caminho de dados. Para a configuração, utilizam interfaces de linha de comando via SNMP e NetConf.

Li et al. [2012] apresentam uma proposta, mediante utilização de uma implementação baseada em SDN, capaz de implantar extensões nos controladores, nos comutadores e nas estações base. A solução proposta permite criar políticas de alto nível baseadas nos atributos dos assinantes, obter controle fino e em tempo real, realizar

DPI (*Deep Packet Inspection*) e compressão de cabeçalhos nos pacotes trafegados e gerenciar o compartilhamento de recursos das estações base. Os autores apresentaram simulações que indicam a possibilidade de melhorias nas redes celulares.

3.3.3 Monitoramento de redes sem fio

Para o bom gerenciamento das redes é necessário conhecer como uma rede se comporta durante sua operação. Os protocolos IEEE 802.11k e IEEE 802.11v são exemplos de iniciativas da indústria de responder a esta demanda. Encontramos na literatura abordagens de monitoramento utilizando SDN para redes sem fio. Nesta seção apresentamos duas propostas: DAIR e *Jigsaw*.

O DAIR (*Dense Array of Inexpensive Radios*), proposto por Bahl et al. [2006], é uma solução de monitoramento de rede para identificar *rogue APs*. Os autores utilizam, para montagem da rede de monitoramento, a infraestrutura de *desktops* existente. Cada computador recebe a instalação de adaptadores USB para redes sem fio IEEE 802.11 e de um agente denominado AirMonitor. O DAIR utiliza um servidor para armazenamento e processamentos dos dados de monitoramento. Apesar dos autores terem mostrado a possibilidade da implementação, esta proposta não abrange a interação entre a rede sem fio e o monitoramento, desta forma não é evidenciada a possibilidade de ações em resposta a atividades não desejadas ou fora do normal observadas pelo monitoramento.

Em *Jigsaw*, Cheng et al. [2006] apresentam um ambiente de monitoramento em ambiente sem fio IEEE 802.11 que utiliza diversos monitores capazes de fornecer uma visão única e global do funcionamento da rede. Os monitores (pontos de acesso sem fio especializados) devem trabalhar em conjunto para conseguir uma visão global do tráfego da rede sem fio. Com isto surge a necessidade de um controlador, nos moldes do SDN, capaz de processar e de consolidar todo o tráfego. A utilização de um controlador centralizado permite ao *Jigsaw* reconhecer e recompor todo o tráfego, ao capturar pacotes em diversos pontos de acesso e agrupá-los em uma visão global da rede. O monitoramento permite estimar o estado dos diferentes canais de rede sem fio, como por exemplo, identificar pontos de acesso em canais interferentes, a carga em cada canal etc. O *Jigsaw* permite caracterizar os enlaces, como por exemplo, os atrasos, as taxas de perda ou a estabilidade do canal. Permite, ainda, descobrir a topologia da rede sem fio, inclusive com a identificação de pontos de acesso próximos pertencentes ou não ao ESS. No caso do *Jigsaw*, o objetivo principal é o monitoramento de rede. Desta forma esta atividade é feita em paralelo com a operação da rede, ou seja, existe uma rede de monitoramento especializada. A unificação destas duas atividades permitiria, por

exemplo, a alteração do papel de um nó de monitoramento para um ponto de acesso, no caso de falha, para garantir cobertura em uma área.

Na seção a seguir apresentamos uma classificação e comparação entre as propostas mostradas neste capítulo, as controladoras comerciais e a nossa proposta.

3.4 Classificação do gerenciamento baseado em SDN

Para compararmos as controladoras comerciais, as propostas apresentadas neste capítulo e a nossa arquitetura, consideramos um conjunto de aspectos que relacionamos a seguir. Apresentamos na tabela 3.2 os resultados desta comparação. Os termos indicados em negrito na relação abaixo são utilizados para identificar estas características na tabela.

- Separação do plano de dados do plano de controle (**SEP**): Indicaremos se há separação entre o sistema que toma as decisões sobre o encaminhamento do tráfego (plano de controle) e o sistema subjacente que efetivamente realiza o encaminhamento da origem para o destino do tráfego (plano de dados) indicadas pelas palavras **sim**/**não** para representar ou não a existência da separação dos planos.

- Autenticação, autorização e auditoria (**AAA**): Uma rede pode suportar protocolos que permitam verificar a identidade digital do usuário (autenticação), garantir que um usuário autenticado tenha acesso somente aos recursos autorizados (autorização) e pode prover a coleta de informações sobre o uso dos recursos do sistema pelos seus usuários (auditoria). Indicamos por **sim** se a proposta considera aspectos de AAA ou por **não** se não considera.

- Qualidade de serviço (**QoS**): Consideramos que o administrador da rede pode possuir mecanismos que o permitam alterar as condições de operação da rede dinamicamente. As redes podem ser provisionadas para operar com um determinado nível de qualidade de serviço capaz de garantir uma qualidade mínima para a experiência percebida pelo usuário na utilização de serviços de rede. Esta qualidade está ligada à arquitetura de recursos da rede, ao desempenho dos enlaces e ao controle exercido sobre os fluxos de dados. Se a proposta considera aspectos de QoS, indicamos por **sim**, caso contrário por **não**.

- Mobilidade (**MOB**): Os usuários podem, principalmente em uma rede densa, migrar de um ponto de acesso para outro ao se movimentarem. Um controlador

central pode acelerar o processo de *handoff* reduzindo o tempo de autenticação, uma vez que tem conhecimento global da rede. Além disto este controlador pode iniciar um processo de *handoff*, promovendo a migração de usuários de um ponto de acesso para outro em função de balanceamento de carga ou controle de topologia. Se a proposta avaliada considera aspectos de QoS, indicamos por **sim**, caso contrário por **não**.

- Monitoramento de tráfego (**MON**): O conhecimento das características do tráfego de rede sem fio permite implementar melhorias de desempenho e segurança, mediante a obtenção de uma visão global e sincronizada de toda a atividade nas camadas física, de enlace, de rede e de transporte. Para isto é necessário o estabelecimento de sensores de rede sem fio capazes de capturar o tráfego (parcial ou completamente), e de servidores capazes de analisar de forma centralizada ou não os dados coletados. Estes sensores podem ser exclusivos (**EXC**) ou não exclusivos (**N**), no sentido destes serem dispositivos completamente dedicados à atividade de monitoramento ou não, executando também atividades relacionados ao encaminhamento de tráfego dos clientes da rede sem fio. Os dados coletados podem ser totais (**TOT**) ou parciais (**PARC**), no sentido de que todo o pacote é recuperado ou somente algumas partes ou estatísticas são recuperadas.

- Controle de encaminhamento (**ENCA**): Os equipamentos de rede (comutadores e roteadores sem fio) encaminham tráfego entre suas portas. O controle deste encaminhamento pode ser baseado em processos tradicionais de comutação ou o controle do encaminhamento pode ocorrer mediante utilização de regras e ações com uma granularidade fina (**FINA**) ou grossa (**GROS**). Ações de granularidade fina atuam sobre uma combinação de campos do cabeçalho do pacote a ser processado pelo dispositivo, enquanto que com granularidade grossa atuam somente nos endereços da camada de enlace e/ou da camada de rede.

- Redes atendidas (**REDE**): A abordagem de redes definidas por software pode ser aplicada às redes cabeadas, como as redes **Ethernet**, e/ou às redes sem fio. Neste capítulo vimos que existem abordagens que apenas foram implementadas para redes **3GPP**, **LTE**, **Wi-Fi** (802.11), **WiMax** ou uma combinação destas.

- Presença de agente no cliente ou ponto de acesso (**AG**): As propostas apresentadas nesta seção efetuaram alterações no funcionamento de módulos da camada de enlace de modo a obter acesso à informações de gerenciamento da rede sem fio. Esta alteração pode ser feita somente no ponto de acesso (**AP**) ou na **estação base** ou no cliente da rede sem fio (**C**).

- Controle de parâmetros da rede sem fio (**PARAM**): O controle em uma rede sem fio é mais complexo que na rede cabeada. Existem muito mais parâmetros a se monitorar e configurar, como por exemplo o canal de operação, a potência de transmissão, intervalo do *beacon*, parâmetros de contenção etc. A possibilidade do controlador avaliar e atuar sobre estes parâmetros permite que a rede sem fio possa ser melhor configurada para atender aos requisitos de usuário. Apresentamos somente se a proposta considera ou não estes parâmetros, sem nos preocuparmos com quais e quantos parâmetros são controlados. Se a proposta avaliada permite o controle de alguns parâmetros da rede sem fio, indicamos por **sim**, caso contrário por **não**.

A tabela 3.2 possibilita a comparação entre as propostas apresentadas nesta seção e os controladores comerciais. Acrescentamos ainda as características que esperamos que a arquitetura proposta nesta dissertação possua. Vemos que a arquitetura do *Ethanol*, quando completamente implementada, permitirá abranger diversas das características enumeradas. Isto porque que nem todas as características enumeradas serão prototipadas durante a dissertação. A proposta do *Ethanol*, nesta dissertação, limitar-se-á ao ambiente de redes sem fio IEEE 802.11 no modo infraestrutura. Em trabalhos futuros, esta arquitetura poderá cobrir outras variações das redes IEEE 802.11 como por exemplo as redes sem fio em malha.

3.5 Conclusão

Com objetivo de mostrar que uma abordagem SDN gera benefícios para uma rede sem fio, sugerimos uma arquitetura SDN para estas redes, com um conjunto limitado de operações de controle. Denominamos esta arquitetura *Ethanol*, em referência à proposta precursora do SDN - o *Ethane*. Usando esta arquitetura podemos efetuar o controle de redes sem fio. A seguir, no capítulo 4, apresentamos a arquitetura proposta e mostramos no capítulo seguinte como foi feita a implementação (parcial) do modelo proposto. No capítulo 5, propomos estudos de caso, validados mediante testes quantitativos de desempenho dos protótipos desenvolvidos.

Proposta	SEP	AAA	QoS	MOB	MON	ENCA	REDE	AG	PARAM
Controladores comerciais	sim[V]	sim	sim	sim	EXC-PARC	GROS	WiMax, Wi-Fi	AP***	sim
Ethane	sim	sim	-	-	-	FINA	Ethernet	-	-
CAPS	sim	não	rede com fio	-	-	FINA	Ethernet	AP	não
CAPWAP	sim	sim	sim	sim	-	FINA	Wi-Fi**	AP	-
CloudIQ	sim	não	sim	sim	-	FINA	3GPP LTE	estações base	sim
CloudMAC	sim	sim	sim	sim	possível	FINA	Wi-Fi**	AP	sim
CoAP	sim	não	não	não	N-PARC	não[IV]	Wi-Fi	AP	sim
CROWD	sim	sim	sim	sim	-	FINA	3GPP LTE, Wi-Fi	estações base, AP	sim
DAIR	sim	não	-	-	EXC-TOT	-	Wi-Fi	clientes	não
EmPOWER	sim	não	não	sim	N-PARC	FINA	3GPP LTE, Wi-Fi	estações base, AP	sim
Jigsaw	sim	não	não	-	EXC-TOT	-	Wi-Fi	AP	não
MobileFlow	sim	não	sim	sim	-	FINA	3GPP LTE	estações base	sim
Odin	sim	sim	-	sim	-	-	Wi-Fi	AP	sim
OpenRF	sim	não	sim	-	-	-	Wi-Fi	AP	sim
OpenRoads	sim	sim	-	-	-	-	WiMax, Wi-Fi	AP	não
SoftCell	sim	não	sim	sim	-	FINA	LTE	estações base, comutadores	sim
SoftRAN	sim	sim	-	sim	-	-	LTE	AP	sim
Ethanol	sim	sim	sim	sim	N-PARC	FINA	Ethernet, Wi-Fi	AP	sim

Notas:

* Os itens indicados com "-" ocorrem quando não podemos afirmar que a característica não esteja presente, por não ser citada diretamente no artigo, ou quando este item não se aplica a arquitetura.

** Em teoria, os conceitos do CAPWAP e CloudMAC podem ser aplicados a qualquer tipo de rede sem fio, porém ambos atualmente estão definidos para IEEE 802.11.

*** Nos controladores comerciais os pontos de acesso sem fio possuem APIs proprietárias que permitem ao fabricante obter informações sobre o comportamento da rede sem fio.

IV O CoAP não trata do encaminhamento de tráfego.

V Podemos considerar que existe uma separação entre os planos de controle e de dados, utilizando uma interface proprietária, na maioria das soluções com controladores comerciais.

Tabela 3.2: Comparação entre as propostas

Característica	Aruba	Cisco	Meraki	Extreme	HP
Alta disponibilidade	1+1	típico 1+1	1+1	1+1	1+1; N+1; N+N
Formato	VM ou dispositivo	VM ou dispositivo	Nuvem	VM ou dispositivo	VM, dispositivo ou nuvem
QoS	Toda a linha	Toda a linha	Toda a linha	Toda a linha	Toda a linha
Mobilidade	L2/L3	L2 / L3***	L2	L2/L3	
Criação de ACL	Toda a linha	Toda a linha	Toda a linha	Toda a linha + GUI	Toda a linha
CAC	Sim	Sim	Sim	Sim	Sim
Gerenciamento de recursos de rádio	Sim	Sim	Sim	Sim	Sim
Firewall *stateless*	L4 / L3 / L2	L4 / L3 / L2	L4 / L3 / L2	L4 / L3 / L2	L4 / L3 / L2
Detecção de *rogue* AP	Sim	Sim	Sim	Sim	Sim
AAA	Sim*	Sim*	Sim**	Sim*	Sim*
Monitoramento RF	AP[V]	AP[IV]	Sim	AP[V]	AP[V]
Suporte a IEEE 802.11 infraestrutura/malha	Sim	Sim	Sim	Sim	Sim
Atualização firmware	APs proprietários	APs proprietários	APs proprietários	APs proprietários	APs proprietários
Suporte a AP	propr/licenciados	proprietário	proprietário	proprietário	proprietário
Suporte a CAPWAP	não	sim	não	não	não
Suporte IEEE 802.11ac	*indoor*/*outdoor*	*indoor*/*outdoor*	Somente *indoor*	*indoor*/*outdoor*	Somente *indoor*
LAG	exceto VM	exceto VM/linha Flex	não	exceto VM	exceto VM
Controle de WLANs	Máx.:16/AP	Máx:512 - 16/AP	Máx.:16/AP	Máx.:16/AP	Máx: 512
Número máximo de estações	Máx: 32.768 (mod. 7240)	Máx: 64 mil (mod. 7500 e 8500)	Não informado	Máx: 16.384 (mod. C5210)	Máx: 30.000 (mod. 870)
Número máximo de APs gerenciados	Máx: 2048 (mod. 7240)	Máx: 6000 (mod. 7500 e 8500) Típico: <25 APs	Máx: 10.000	Máx: 2000 (mod. C5210)	Máx: 1.536 (mod. 870)
Garantia do software	90 dias (gratuito)	90 dias (gratuito)	Duração do contrato	Contrato mín.1 ano	Contrato mín.1 ano
Suporte	Contrato	Contrato	Contrato	Contrato	3 anos 24x7

Fontes: Aruba Networks [2015]; Cisco Systems [2015a]; Meraki Networks [2015]; Extreme Networks [2015]; HP Networks [2015]; `https://docs.meraki.com/display/MR/Wireless+LAN`; `http://h17007.www1.hp.com/br/pt/networking/solutions/wlan/index.aspx`;

Notas:

* Permite autenticação em banco de dados de usuários local ou via RADIUS. Permite gravação de log utilizando syslog ou em arquivo texto local.

** Permite autenticação via em banco de dados de usuários local, RADIUS, *Facebook Login* ou *Google Authentication for Sign-on.*

*** Controlador virtual e linha Flex não suportam mobilidade IP.

IV Não está presente no controlador virtual e linha Flex.

V Depende do ponto de acesso para funcionar simultaneamente como AP e monitor.

Tabela 3.3: Comparação entre as controladoras comerciais

Capítulo 4

Ethanol: Uma plataforma SDN para redes Wi-Fi

Neste capítulo apresentamos o *Ethanol*. O *Ethanol* é uma plataforma SDN para redes IEEE 802.11 densas, como, por exemplo, em uma rede com muitos APs e clientes - como um campus, uma rede corporativa, uma rede de coisas em uma casa. O *Ethanol* permite o desenvolvimento de software de controle personalizado, permitindo que administradores de rede habilitem serviços que melhor atendam às suas necessidades e de seus usuários. Além de encaminhamento, o controlador também pode controlar a mobilidade das estações, a autenticação de usuários, a criação de uma rede virtual, o *QoS* e, até mesmo, a localização do usuário. O *Ethanol* é uma arquitetura aberta que permite a criação de algoritmos de controle sob medida para as necessidades dos usuários de redes locais sem fio.

O *Ethanol* adota as seguintes diretrizes de projeto: (i) suportar as redes locais sem fio definidas pelo padrão IEEE 802.11, bem como as redes Ethernet; (ii) não exigir mudanças nas estações (os dados que precisem ser coletados a partir das estações são feitos com base no suporte oferecido pela família de protocolos IEEE 802.11); e (iii) fornecer APIs para a mobilidade dos nós, para a virtualização dos pontos de acesso, para a segurança em rede sem fio e para *QoS*.

A arquitetura proposta adota a abordagem SDN para separação do plano de dados do plano de controle focado em redes locais sem fio aderentes ao padrão IEEE 802.11. O *Ethanol* fornece mecanismos para configuração e gerenciamento de redes Wi-Fi, abrangendo aspectos relacionados com a configuração, a qualidade de serviço, a mobilidade e a segurança. Ao realizar modificações somente nos pontos de acesso, utiliza as mensagens de gerenciamento da família IEEE 802.11 para obter informações e gerenciar as estações sem fio.

Iremos na seção 4.1, a seguir, descrever a arquitetura proposta para o *Ethanol*. Nas seções seguintes apresentaremos uma visão condensada das operações possíveis com a arquitetura proposta, seus potenciais usos e vantagens. A implementação da arquitetura é descrita na seção 4.3.

4.1 Arquitetura do *Ethanol*

A arquitetura *Ethanol*, como mostrado na figura 4.1, é composta por dois tipos de dispositivos: o controlador *Ethanol* e os roteadores *Ethanol*. O controlador é executado em um servidor na rede com fio ou virtualizado na nuvem. Os pontos de acesso *Ethanol* são roteadores sem fio IEEE 802.11, que são modificados para executar código da arquitetura *Ethanol*.

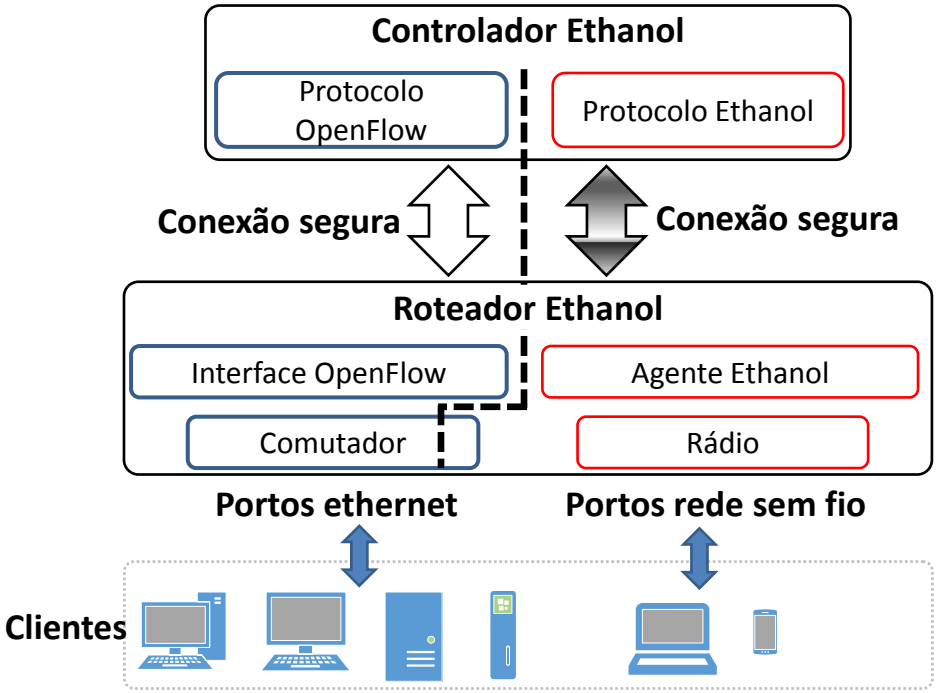


Figura 4.1: Arquitetura da solução com *Ethanol*

Um ponto de acesso *Ethanol* possui duas partes: os componentes de rede Ethernet e os componentes de rede sem fio. Um componente de rede Ethernet é um elemento de comutação configurável que suporta o protocolo OpenFlow. Como o OpenFlow não fornece uma interface para os parâmetros de rede sem fio e para configuração das filas de prioridades utilizadas para *QoS*, o agente *Ethanol* instalado nos roteadores adiciona essas funcionalidades. Este agente usa uma API adicional que, além de permitir gerenciar as filas de encaminhamento para *QoS*, é capaz de controlar os parâmetros de

rede sem fio. Este agente recebe os comandos do controlador *Ethanol* por meio de um canal seguro usando XML-RPC.

O *Ethanol* não necessita de modificações nos clientes, mas emprega os protocolos IEEE para obter informações sobre as estações e alterar seu comportamento, como detalhado na seção 3.1.2. Para que o *Ethanol* seja implementado não é necessário que os dispositivos possuam suporte completo aos protocolos indicados, contudo quanto maior o suporte oferecido, maior será a gama de operações que poderão ser feitas pelo *Ethanol*.

A arquitetura do *Ethanol* acrescenta diversas operações não suportadas pelo protocolo OpenFlow. Utilizando os novos recursos propostos, uma rede SDN com o *Ethanol* pode gerenciar a qualidade dos enlaces, pode obter dados das estações, permite habilitar a virtualização de APs, pode controlar o fluxo de dados etc, como mostraremos na seção 4.3.

4.2 Implementação do *Ethanol*

O controlador *Ethanol* é uma extensão do controlador POX (`http://www.noxrepo.org/category/controllers/pox/`). Em função disto, o *Ethanol* é compatível com a versão 1.0 do OpenFlow. Para que outra versão do OpenFlow possa ser utilizada junto com o *Ethanol*, o POX não poderá ser utilizado, implicando o readequação do nosso código.

Optamos nesta dissertação por utilizar o POX. Esta decisão ocorreu em função dos experimentos iniciais com o roteador WRT54G que utilizava OpenWRT (`https://openwrt.org/`) com Pantou (`http://archive.openflow.org/wk/index.php/Pantou_:_OpenFlow_1.0_for_OpenWRT`). O Pantou é uma implementação do OpenFlow versão 1.0 para a plataforma OpenWRT. Para comunicar com um dispositivo assim configurado, precisávamos de um controlador compatível com esta versão e que fosse fácil de utilizar. Daí a escolha do POX. Ao migrarmos para uma plataforma em Linux com *OpenvSwitch*, consideramos manter o POX em função da quantidade de código que deveria ser alterada para compatibilizar com outro controlador.

A arquitetura do *Ethanol* utiliza dois protocolos de comunicação entre o controlador e os roteadores *Ethanol* - o OpenFlow e o protocolo *Ethanol*. A figura 4.1 mostra um esquema de como é feita a comunicação no ambiente da solução proposta. Podemos notar que os dois protocolos de comunicação coexistem lado a lado. O protocolo de controle das interfaces Ethernet é compatível com OpenFlow versão 1.0. A utilização de uma versão mais nova do OpenFlow pode ser feita para o *Ethanol*, contudo isto

não pode ser feito utilizando as bibliotecas do POX, pois a versão "Carp"[1] (versão mais atual do POX) suporta somente a especificação 1.0 do OpenFlow, com suporte a algumas funcionalidades da versão 1.1 e extensões para Nicira/*OpenvSwitch*.

O desenvolvimento do *Ethanol* foi feito utilizando as linguagens Python e C. No controlador, as aplicações são desenvolvidas na linguagem Python para que sejam compatíveis com o controlador POX (OpenFlow). Já no lado dos roteadores *Ethanol*, as aplicações são desenvolvidas na linguagem C, em função da necessidade de alterar o *hostapd*. O *hostapd* é a implementação em espaço de usuário para Linux do ponto de acesso IEEE 802.11 e do autenticador IEEE802.1X/WPA/WPA2/EAP/RADIUS. Este módulo está escrito em C. Mais informações podem ser obtidas em `https://wireless.wiki.kernel.org/en/users/documentation/hostapd`.

Era necessário portanto que os programas nestas duas linguagens conseguissem trocar mensagens entre si. A comunicação entre o controlador e o comutador é feita por dois protocolos: (1) o *protocolo OpenFlow* é utilizado sem modificações para controlar os portos do comutador e sua tabela de encaminhamento e (2) o *protocolo Ethanol* se encarrega da comunicação das informações relacionadas às especificidades da rede sem fio e para configuração das filas de prioridade do comutador.

Para implementar a comunicação cliente-servidor do protocolo *Ethanol*, optamos por utilizar o XML-RPC. A decisão de utilizar o XML-RPC deu-se pela disponibilidade de bibliotecas para os ambientes C e Python, utilizados em nossos protótipos, e pela facilidade de utilização destas bibliotecas. Como sugestão para otimização da comunicação pode ser adotada uma abordagem similar à do OpenFlow, utilizando *socket* SSL. Uma ampliação do protocolo OpenFlow também pode ser feita, estendendo o protocolo para suportar estas novas mensagens do *Ethanol*.

Para estabelecer a comunicação entre o controlador *Ethanol* e seus agentes foi necessário desenvolver um canal seguro de comunicação entre os dois equipamentos, como esquematizado na figura 4.2. Como o controlador *Ethanol* deve enviar comandos para o cliente *Ethanol*, é necessário que no cliente *Ethanol* esteja rodando um processo servidor, capaz de receber estas mensagens, processá-las e responder ao controlador. Da mesma forma, o controlador *Ethanol* deve responder a solicitações dos clientes, como por exemplo, quando uma estação tenta conectar na rede sem fio, o cliente *Ethanol* pergunta ao controlador se esta conexão deve ser aceita ou não. Portanto foi necessário criar uma aplicação servidor e uma aplicação cliente para cada um dos equipamentos.

Desta forma utilizamos em C a biblioteca *XML-RPC para C/C++* (`http://xmlrpc-c.sourceforge.net/`). Esta biblioteca fornece para o cliente suporte a

[1] `https://github.com/noxrepo/pox`.

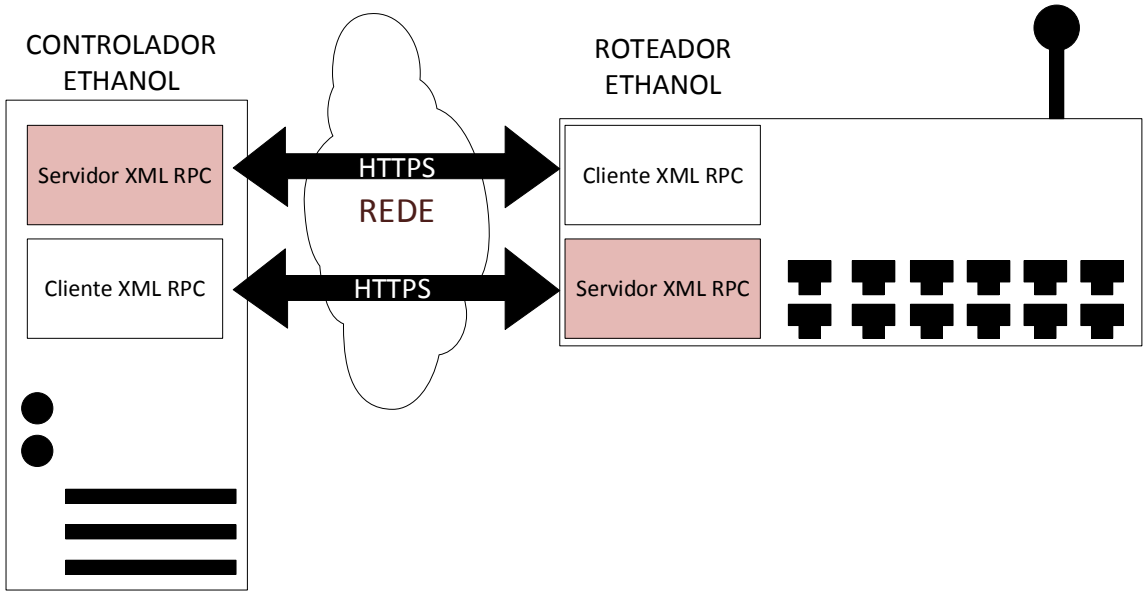


Figura 4.2: Esquema de comunicação cliente servidor utilizando XML-RPC

HTTPS via *Curl* (`http://curl.haxx.se/`). O servidor no roteador foi desenvolvido utilizando XML-RPC com *Abyss Server* (`http://abyss.sourceforge.net/`). Conforme os desenvolvedores do *xmlrpc-c*, uma alternativa simples para criar uma conexão segura é o uso do *stunnel* (`https://www.stunnel.org/`) que permite implementação para o cliente e para o servidor, uma vez que é baseado em encaminhamento de portas. Em função de não existirem exemplos de desenvolvimento usando *Abyss* com HTTPS, no servidor foi utilizado o *stunnel*. O Python, a partir da versão 2.2, dispõe da biblioteca *xmlrpclib* que implementa o protocolo XML-RPC utilizando HTTP ou HTTPS. Este recurso é muito simples e foi utilizado nos módulos em Python do nosso controlador.

A comunicação no protocolo OpenFlow é segura e utiliza SSL no porto TCP padrão 6653 (Foundation [2013]). Já o protocolo de comunicação do *Ethanol* utiliza-se de chamadas XML-RPC, usando HTTPS como transporte seguro, pelo porto TCP 22222. O porto de conexão no módulo em XML-RPC pode ser configurado, como no OpenFlow.

4.2.1 Plataformas de hardware do *Ethanol*

O roteador *Ethanol* foi desenvolvido utilizando duas plataformas - OpenWRT e Linux. A primeira versão foi desenvolvida para os roteadores Linksys modelo WRT54G e TP-Link modelo TL-WR2543ND. Nos nossos experimentos iniciais identificamos algumas dificuldades encontradas na depuração dos programas para plataformas OpenWRT e

na instalação do *firmware* nos dispositivos. Isto influenciava o tempo necessário para a prototipação. Desta forma, adotamos o desenvolvimento em computador com sistema operacional Linux nos casos apresentados no capítulo 5.

Tendo o protótipo desenvolvido em ambiente linux é possível instalar o *Ethanol* em roteadores de mercado. Estes dispositivos podem ser instalados com OpenWRT que fornece um amplo suporte às operações de rede. A principal dependência que encontramos diz respeito ao *kernel* do OpenWRT suportado por cada equipamento, o que define quais padrões IEEE 802.11 estarão disponíveis no equipamento, bem como outros recursos como a versão do *Openvswitch* disponível para a plataforma. Além disto, cada plataforma possui restrições diferentes de processamento e de memória RAM (*Random Access Memory*) e memória FLASH, o que reduz as funcionalidades que podem ser instaladas sem desenvolvimento adicional.

O agente *Ethanol* foi incorporado ao módulo *hostapd* do Linux nos roteadores. Foram alterados os procedimentos relacionados ao controle das mensagens de gerenciamento e tratamento de *beacon* do *hostapd*. Desta forma ao ser carregado o módulo *hostapd*, a nossa implementação também é executada. Optamos por não alterar o arquivo de configuração do *hostapd*. Desta forma criamos um arquivo de configuração no formato "INI" (`http://fileinfo.com/extension/conf`) que é lido durante a carga do módulo. Este arquivo em formato texto mantém as configurações estáticas do agente, como por exemplo porto do servidor XML-RPC a ser executado no agente. Uma opção neste arquivo desabilita toda a implementação feita para o *Ethanol* fazendo com que o *hostapd* funcione com sua implementação original.

Para os experimentos do capítulo 5, o controlador *Ethanol* foi instalado em um computador Linux Intel x64 com 2 GB RAM, rodando Ubuntu versão 14.04.2 LTS, com Python versão 2.8 e POX *Carp*. Os roteadores *Ethanol* foram instalados em computadores Linux Intel x64, 2 GB RAM, Ubuntu versão 14.04.2 LTS, *OpenvSwitch* versão 2.0.2 e *hostapd* versão 2.1. Em cada um destes computadores foram instalados um adaptador de rede sem fio com chipset AR9170 802.11n e um conjunto de placas de rede PCI, que totalizam 3 ou 5 portos 10/100BaseTX, dependendo da configuração e quantidade de placas Ethernet PCI utilizadas.

4.2.2 Roteador *Ethanol* empregando XML-RPC

Identificamos durante os experimentos dos estudos de caso que a biblioteca *xmlrpc-c* gera uma grande sobrecarga de tamanho das implementação do protótipo (tamanho do arquivo *hostapd*) e no tamanho das mensagens. Tomemos como exemplo, a mensagem "*hello*" que é a mensagem que informa a conexão do agente para o controlador.

Ao ser codificada como XML, a mensagem recebe um conjunto de *tags* e atributos adicionais. Na coluna da direita, abaixo, podemos ver o exemplo de uma mensagem "*hello*" com um único endereço MAC para a interface de rede sem fio. A mensagem em formato xml possui 773 bytes, ficando mais de 10 vezes maior que a mesma mensagem em formato binário, que tem 61 bytes.

```
struct hello_s {
    char* p_version;
    int m_type;
    int m_size;
    int m_id;
    int numMACs;
    char * listMACaddress;
    int serverPort;
};
```

```
4I4IPOST /RPC2 HTTP/1.1
Host: localhost:22222
Accept: */*
Content-Type: text/xml
User-Agent: Xmlrpc-c/1.25.30 Curl/7.35.0
Content-Length: 672
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>method</methodName>
  <params>
    <param><value><struct>
      <member><name>Versao</name>
        <value><string>1.0</string></value>
      </member>
        <member><name>Tipo</name>
      <value><i4>0</i4></value>
      </member>
        <member><name>Tamanho</name>
      <value><i4>0</i4></value>
      </member>
      <member><name>ID</name>
        <value><i4>0</i4></value>
      </member>
      <member><name>IPS</name>
        <value><string>192.168.10.20;192.168.10.8;
        </string></value>
      </member>
      <member><name>MACS</name>
        <value><string>44:33:4c:07:49:9b;</string></value>
      </member>
      <member><name>PORT</name>
        <value><i4>22223</i4></value>
      </member>
    </struct></value></param>
  </params>
</methodCall>
```

Em seguida analisamos o tamanho do código gerado. Mostramos na tabela 4.1 que o módulo *hostapd* original do Linux, na versão 2.1, tem um tamanho de aproximadamente 992 kB (1016144 bytes). Enquanto o mesmo software com as modificações para os estudos de caso apresentados na dissertação ficou com aproximadamente 5,1 MB (5382427 bytes). Computamos a quantidade de bytes adicionadas pelas modificações realizadas no código original do hostapd e o acréscimo realizado pela biblioteca xmlrpc-c. Observamos que a maior quantidade é devida a esta biblioteca (cerca de 78,4% do

código acrescentado).

Software	**Sem alteração (em kB)**	**Com alteração (em kB)**
Módulos do *Ethanol*	0	144,8
Biblioteca xmlrpc-c	0	4119,1
Hostapd	**992,3**	**5256,3**

Tabela 4.1: Acréscimo de tamanho ao módulo *hostapd*

Este acréscimo de tamanho no *hostapd* inviabiliza sua instalação nos roteadores Linksys modelo WRT54G, que possui menos de 1 MB livre, após a instalação mínima do OpenWRT. Este roteador não apresenta possibilidade de expansão de memória. Já o roteador TP-Link modelo TL-WR2543ND possui menos de 4 MB livres após a instalação mínima do OpenWRT. Contudo o TL-WR2543ND possui uma entrada USB, que permite instalação de um pendrive, onde pode ser colocado o módulo *hostapd*. Desta forma abre-se a possibilidade de utilizar este armazenamento externo para conter a versão modificada do *hostapd*. Deixamos esta implementação para trabalhos futuros.

4.3 Operações suportadas pelo *Ethanol*

A API do *Ethanol* é projetada segundo uma abordagem orientada a objetos. Esta abordagem trabalha com entidades que possuem propriedades, executam ações por meio de seus métodos e são acionadas por eventos, como por exemplo na chegada de um pacote a um comutador. Estas entidades, mostradas na figura 4.3, representam objetos físicos ou virtuais que podem ser configurados ou observados. Um AP e um enlace são exemplos de uma entidade física e uma entidade virtual, respectivamente.

As entidades possuem propriedades observáveis e/ou configuráveis, tal como o ESSID (*Extended Service Set Identification*), a potência do sinal transmitido ou recebido, o número de clientes associados em um AP etc. As propriedades são consultadas pelo controlador usando métodos *get*/*set*, como por exemplo, $getNumClients()$ ou $setSSID(ssid)$. Estes métodos de leitura e escrita das propriedades não são mostrados no modelo da figura 4.3. Por exemplo, a classe $Network$ possui uma propriedade $SSID$, que é de leitura e escrita, portanto esta classe possui um método $getSSID()$ e um método $setSSID(ssid)$ que não são mostrados. Existem ainda métodos originados dos relacionamentos entre as classes, que também não são mostrados. Por exemplo, um objeto da classe $Network$ pode estar relacionado a diversos objetos da classe $VirtualAccessPoint$. Este relacionamento indica quais pontos de acesso pertencem a uma determinada rede. Dessa forma, existe um método $getVAPs()$ na classe

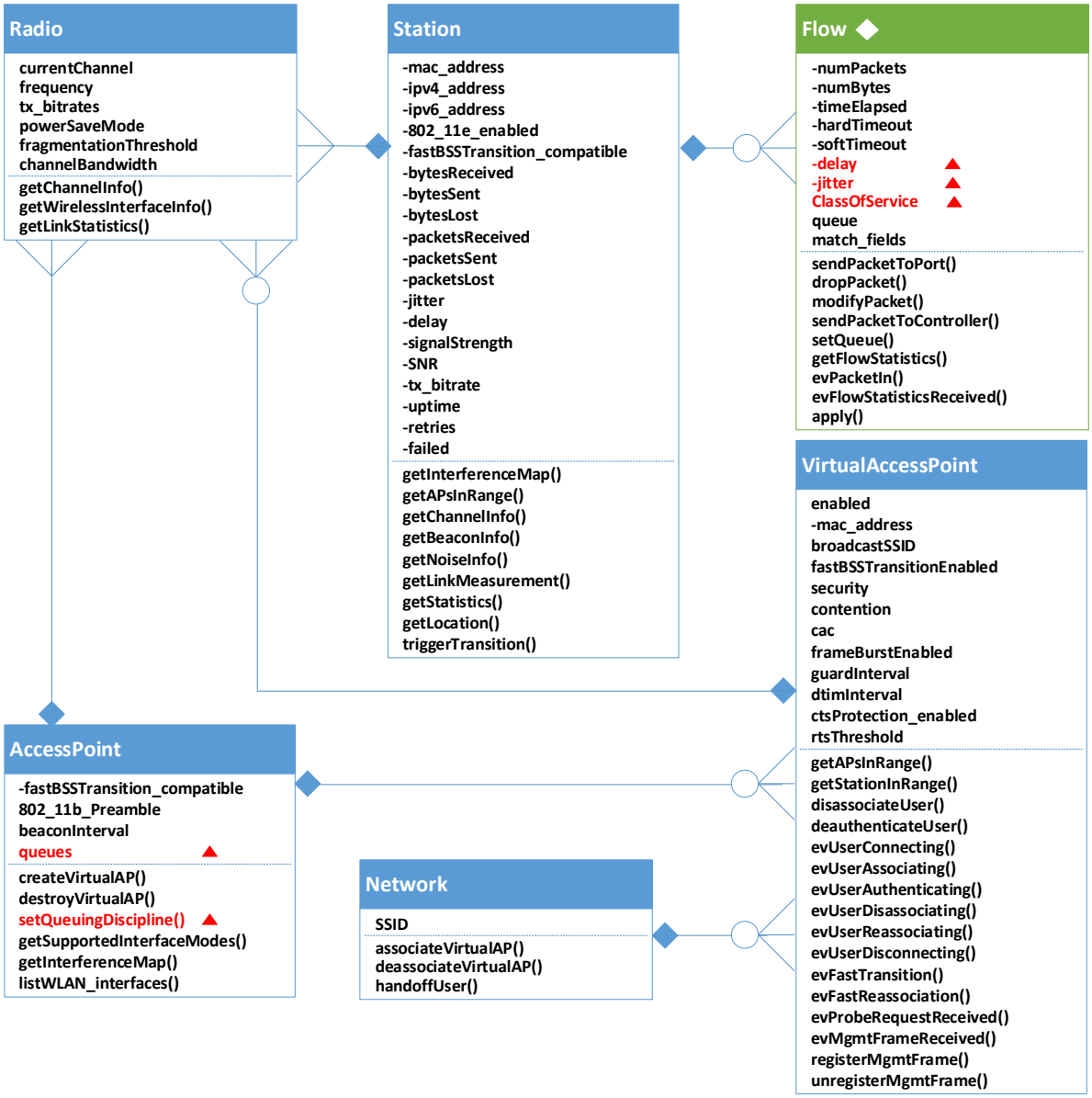


Figura 4.3: Modelo da API do *Ethanol*

Network que retorna os pontos de acesso de uma determinada rede. Este método também não é explicitado no modelo da figura.

As entidades podem ter eventos, que geram chamadas para o controlador, de modo que este possa responder a situações representadas nesses eventos de forma apropriada. Por exemplo, quando uma estação sem fio deseja conectar a um ponto de acesso, um evento é gerado, permitindo ao controlador definir se esta associação será autorizada ou não. Nesta seção apresentaremos uma visão condensada do modelo. Uma descrição detalhada do modelo de classes pode ser vista no Anexo A.

A entidade em verde, marcada com um losango branco, corresponde à implementação atual do OpenFlow, enquanto os métodos e atributos destacados em vermelho e marcados com um pequeno triângulo correspondem às adições do modelo à implementação de controle de fluxos do OpenFlow.

Cada classe é apresentada no diagrama com duas partes separadas por uma linha tracejada. Na parte superior estão os atributos de classe e, na parte inferior, estão os métodos e eventos tratados pelos objetos da classe. Os nomes dos eventos iniciam com "**ev**". Os atributos com um símbolo "**-**" à esquerda são atributos que somente podem ser lidos.

Algumas das propriedades e métodos propostos podem não ser viáveis em alguns pontos de acesso de produção, em função de limitações de hardware e/ou software. No entanto, optamos por especificar a arquitetura sem levar em conta as limitações dos equipamentos existentes. Hardwares futuros podem ser desenvolvidos baseados nesta especificação em uma tendência semelhante ao que aconteceu com SDN: suas primeiras especificações foram limitadas às funções implementáveis em hardware existente, contudo agora os fabricantes de equipamentos de redes baseados em SDN estão adaptando seu hardware para operações SDN. Fornecemos nas subseções seguintes uma breve descrição dessas entidades.

O modelo de classes do *Ethanol*, mostrado na figura 4.3 e que trataremos em mais detalhes nas subseções a seguir, foi implementado como classes em Python derivadas da classe *object*. Os atributos foram decorados com *@property*, *@x.setter* e *@x.getter* para permitir que estes fossem acessados como atributos somente leitura ou leitura e escrita. Os objetos criados durante a operação do controlador não possuem mecanismos de persistência implementados na nossa plataforma. A implementação de persistência pode ser feita em Python utilizando, por exemplo, *cPickle*. Os objetos serializados podem ser gravados em disco. Contudo esta característica não é essencial para a dissertação e portanto não foi implementada. Sugerimos a implementação de persistência como um futuro desenvolvimento da plataforma.

4.3.1 *AccessPoint*

Esta entidade representa os dispositivos físicos - os roteadores sem fio. Um *AccessPoint* pode ter um ou mais rádios físicos (como em pontos de acesso com rádios de 2,4 GHz e 5 GHz), representados pela entidade *Radio*, e um ou mais pontos de acesso virtuais em execução (classe *VirtualAccessPoint*). A entidade *AccessPoint* possui três atributos principais: *beaconInterval* (afeta a frequência dos sinais de *beacon*), *fastBSSTransition_compatible* (indica se o ponto de acesso é compatível com o protocolo IEEE

802.11r) e *802_11b_Preamble* (indica se o preâmbulo utilizado é longo ou curto).

Os métodos indicados pelos atributos e métodos com um triângulo são aqueles que fornecem suporte a QoS, permitindo consultar o estado das filas para cada porto do dispositivo, bem como alterar a prioridade das filas e a disciplina de filas utilizada pelo dispositivo (escalonadores de filas no núcleo do sistema operacional). Estes métodos acrescentam a possibilidade de configuração de filas, não suportada pelo OpenFlow sem o uso de comandos externos. Os métodos disponíveis nesta classe permitem a criação e destruição de pontos de acesso virtuais. É possível ainda verificar informações da interface como, por exemplo, quais são as interfaces de rede sem fio e quais os modos suportados por estas interfaces (por exemplo, se infraestrutura ou malha). O método $getInterferenceMap()$ fornece um mapa da interferência, por meio da varredura dos canais suportados pelo ponto de acesso.

4.3.2 *Radio*

Esta entidade configura a interface sem fio (rádio) do ponto de acesso e permite acesso às informações de RF. A classe possui atributos como o canal, a potência do transmissor, as taxas de bits configuradas e o uso de modo de economia de energia. Esta classe possui dois métodos que obtêm informações do enlace, obtidas junto ao módulo *hostapd* do ponto de acesso.

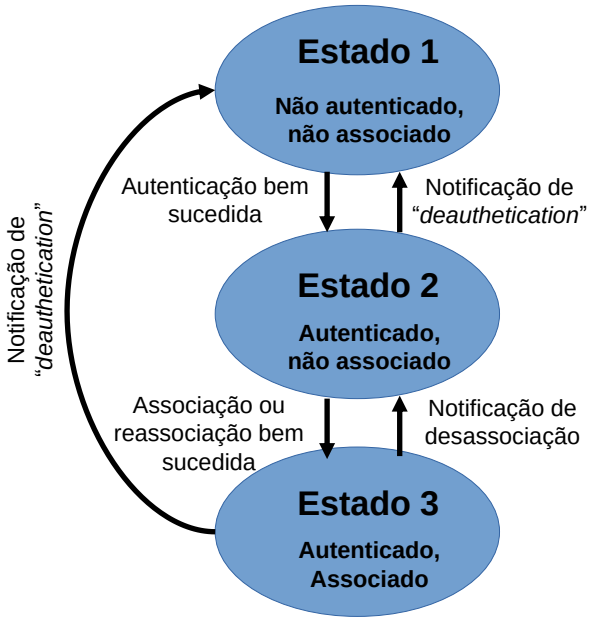
Os objetos desta classe informam as estatísticas de tráfego e outras informações do enlace físico que representam. Estas informações podem ser solicitadas a cada estação conectada, mediante mensagens IEEE 802.11k enviadas às estações, o que permite ao AP solicitar informações sobre o ambiente de rádio. Por meio deste protocolo estão disponíveis relatórios ([IEEE802.11k, 2008, p.3-6]) como "*Channel Load*", "*Noise Histogram*" e "*Link Measurement*".

4.3.3 *VirtualAcesssPoint*

Os pontos de acesso virtuais correspondem às entidades abstratas nas quais os usuários são conectados. Um dispositivo físico pode ter zero ou mais pontos de acesso virtuais. Se for zero, o dispositivo não está fornecendo qualquer serviço de rede. É possível configurar um AP virtual, mas mantê-lo desativado para uso futuro ou inicialização rápida. Se *broadcastSSID* está desativado ($False$), então este ponto de acesso virtual não transmite seu SSID. As estações conectam-se a um ponto de acesso virtual e um grupo destes pontos de acesso virtuais formam uma *Network*.

Esta entidade controla, ainda, os parâmetros de transmissão do MAC, como intervalos de guarda e intervalos DTIM (*Delivery Traffic Indication Message*), o limiar de RTS (sinal de *Request To Send*) e informações sobre as características do enlace (por exemplo, se o "*frame burst*" está ativado). Também expõe os parâmetros de contenção do protocolo IEEE 802.11e (por exemplo, valores máximos e mínimos da janela de contenção e o valor do AIFS) e parâmetros de controle de admissão. Cada AP virtual também armazena seus próprios parâmetros de segurança.

As transições no processo de associação do usuário, representadas pela máquina de estados da figura 4.4, geram um conjunto de eventos para o controlador, como $evUserConnecting()$, $evUserAssociating()$, $evUserAuthenticating()$, $evUserDisassociating()$, $evUserReassociating()$ e $evUserDisconnecting()$. Estes eventos permitem que o controlador aceite ou negue a solicitação em cada etapa do processo de associação. A entidade também possui eventos para responder a solicitações de transição rápida e de reassociação rápida (conforme definido no protocolo IEEE 802.11r) ou para pedidos de sondagem ("*Probe Requests*").



Adaptado de [Gast, 2005, p.112].

Figura 4.4: Processo de associação simplificado

Nossa arquitetura permite um comportamento similar ao CAPWAP (Calhoun et al. [2009]), isto é, é possível enviar ao controlador as mensagens de gerenciamento da rede sem fio. Contudo, utilizando os métodos $registerMgmtFrame()$ e $unregisterMgmtFrame()$, é possível definir quais as mensagens serão encaminhadas

ao controlador e quais serão tratadas no ponto de acesso pelos procedimentos atuais do *hostadp* definidos pelo padrão IEEE 802.11.

4.3.4 *Network*

Esta entidade representa uma rede (conjunto de pontos de acesso virtuais) e seu ESSID. Uma rede pode conter vários $VirtualAccessPoints$. Esta entidade fornece métodos para associar ou desassociar os pontos de acesso virtual à rede, bem como um método para solicitar um *handoff* de um usuário.

O método $handoffUser()$ utiliza os recursos fornecidos pelo protocolo IEEE 802.11r, que introduz um novo conceito de *roaming*, onde o *handshake* inicial com o novo AP é feito antes mesmo do cliente mudar para o novo ponto de acesso. O IEEE 802.11r elimina grande parte da sobrecarga do *handshaking* em *roaming*, reduzindo, assim, os tempos de *handoff* entre APs, proporcionando segurança e QoS. Isso é útil para dispositivos clientes que executam aplicações sensíveis a atrasos, tais como voz e vídeo. Para que o protocolo IEEE 802.11r funcione, é necessário que o protocolo IEEE 802.11k esteja implementado pois a estação utiliza relatórios de varredura (*Scanning Reports*) e de vizinhança (*Neighbor Reports*) para identificar os pontos de acesso.

4.3.5 *Station*

Esta entidade representa uma conexão do usuário (estação) a um ponto de acesso. Todos os dados são solicitados à estação pelo controlador por meio de mensagens conforme definido pelo protocolo IEEE 802.11k. Com estas mensagens temos acesso aos relatórios *Beacon*, *Frame*, *Channel Load*, *Noise Histogram*, *STA Statistic*s, *Location Configuration Information*, *Neighbor Report*, *Link Measurement* e *Transmit Stream/-Category Measurement*. A entidade utiliza diversos métodos que utilizam os protocolos IEEE 802.11k e 801.11v para solicitar estas informações. A lista dos pontos de acesso ao alcance da estação pode ser obtida usando $getAPsInRange()$ - útil para otimizações pré-*handoff* ou durante o processo de associação. Já os valores dos contadores podem ser solicitados usando o método $getStatistics()$, que fornece, por exemplo, o número de quadros/pacotes transmitidos, o número de quadros/pacotes não transmitidos, as tentativas de transmissão etc. $getLocation()$ retorna a localização geográfica utilizando os dados fornecidos pelo relatório *Location Configuration Information*.

A classe contém informações das interfaces de rede da estação, como seus endereços MAC e IP. Podemos ainda requisitar se a estação suporta o protocolo IEEE 802.11e (que habilita QoS no ambiente sem fio). Podem ser solicitadas ainda informações sobre

a conexão entre a estação e o ponto de acesso, tais como o número de bytes/pacotes enviados e recebidos, a força do sinal, o valor do SNR, a taxa de bits, o número de tentativas de transmissão, o valor da perda de pacotes, do atraso e das flutuações do atraso (*jitter*).

Uma estação pode ser solicitada a trocar de ponto de acesso mediante a utilização do método $triggerTransition()$. Este método utiliza o frame "BSS transition management", definido pelo IEEE 802.11v, para que um ponto de acesso requisite que uma estação realize uma transição para outro ponto de acesso. Este método utiliza somente mecanismos fornecidos pelo IEEE 802.11v, portanto caso o controlador precise forçar a desconexão da estação do AP atual ou gerenciar processos de reautenticação, o método $handoffUser()$ da classe $Network$ deverá ser chamado.

4.3.6 *Flow*

Esta entidade origina-se da especificação do OpenFlow, contendo, portanto, os métodos que encapsulam as ações descritas neste protocolo, permitindo acesso aos contadores e recebendo os eventos. Esta entidade possui algumas características adicionais para permitir maior controle do fluxo de dados em uma rede sem fio, como por exemplo o atraso, a taxa de perdas de pacotes e as flutuações do atraso (*jitter*).

Na classe $VirtualAccessPoint$ existem métodos complementares que permitem a configuração das filas de prioridade, indicadas no modelo da figura 4.3 pelos triângulos na cor vermelha.

4.4 Exemplo de uso da arquitetura

Nesta seção apresentaremos um exemplo desenvolvido em Python de como a arquitetura pode ser utilizada para encaminhar um fluxo de dados para uma fila de prioridades. Esta implementação foi utilizada para controle de fluxo na seção 5.2.

Vemos na linha 1 da listagem 4.1 que a classe $Flow$ é importada da biblioteca do *Ethanol*. Para fins de simplificação, não mostramos as linhas de código que não alteramos do comutador *forwarding.l2_learning* fornecido pelo POX. A partir da linha 5 relacionamos como ficou o procedimento *_handle_PacketIn()* reescrito para utilizar os recursos fornecidos pela arquitetura. As linhas 6 e 7 não sofreram modificação. Elas são utilizadas para decodificar o pacote recebido e o porto de origem do pacote, respectivamente.

Nas linhas de 11 a 13, mostramos como um fluxo é descartado. A classe $Flow$ é instanciada com os valores padrão (linha 11). Ela recebe dois parâmetros: *event*

Listagem 4.1: Alteração de _handle_PacketIn para controle de fluxos

```
1  from ethanol.flow import Flow
2
3  ...
4
5  def _handle_PacketIn(self, event):
6    packet = event.parsed
7    self.macToPort[packet.src] = event.port # 1
8
9    if not self.transparent: # 2
10     if packet.type == ethernet.LLDP_TYPE or packet.dst.isBridgeFiltered():
11       flow = new Flow(event, self.connection) # 2a drop
12       flow.dropPacket()
13       flow.apply()
14       return
15   if packet.type == ethernet.ARP_TYPE: # ARP
16     self.do_flood(event, "ARP")
17   elif packet.dst.is_multicast:
18     self.do_flood(event, "Multicast") # 3a
19   else:
20     if packet.dst not in self.macToPort: # 4
21       self.do_flood(event, "%s Port to %s unknown -- flooding" % (timestamp(), packet.dst)) # 4a
22     else:
23       port = self.macToPort[packet.dst]
24
25       flow = new Flow(event, self.connection)
26       flow.hardTimeout = HARD_TIMEOUT
27       flow.softTimeout = SOFT_TIMEOUT
28       if port == event.port: # 5
29         # 5a
30         flow.dropPacket()
31       else:
32       # 6
33         if packet.type == ethernet.IP_TYPE:
34           packet_ip = packet.next
35           # notar que se nao acha uma fila para o endereco, fila retorna None
36           fila = get_queue_by_ip(lista_ip, packet_ip.dstip, packet_ip.srcip)
37           if fila != None:
38             flow.setQueue(queue_id = fila)
39       flow.sendPacketToPort(port = port)
40       flow.apply()
```

e *connection*. Toda vez que um comutador conecta-se ao controlador POX, é criado um objeto da classe *Connection* (`https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-ConnectionObjects`). O objeto *connection* é utilizado para enviar mensagens do controlador para o comutador. Quando o evento *PacketIn* é acionado, em função do recebimento de uma mensagem *OFPT_PACKET_IN* pelo controlador, este chama o procedimento cadastrado para tratar este evento (`https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-PacketIn`). No nosso exemplo, o tratador deste evento é *_handle_PacketIn*. Este tratador recebe como parâmetro *event*, que contém o porto de entrada do pacote no comutador bem como a mensagem decodificada.

Desta forma descartar um pacote é um processo de 3 etapas: (1) instanciar a classe *Flow*, como na linha 11, (2) chamar o método *dropPacket*() para indicar que

o fluxo representado por este pacote deverá ser descartado (linha 12) e (3) executar *apply*() para enviar a mensagem para o comutador, como na linha 13.

A próxima alteração do procedimento *_handle_PacketIn* ocorre nas linhas de 25 a 40. Na linha 25 a classe *Flow* é instanciada para manipular o fluxo de deverá ser encaminhado para as filas de prioridade. Podemos definir os períodos dos temporizadores definidos pelo OpenFlow - *Idle timeout* e *Hard timeout*. O primeiro é alterado usando $softTimeout$, mostrado na linha 27 da listagem, enquanto ao segundo é alterado por $hardTimeout$ (linha 26).

Quando o porto de entrada do pacote é o mesmo do porto de saída, o fluxo é simplesmente descartado. Esta condição é testada na linha 28. O fluxo é programado para ser descartado na linha 30. Esta alteração é efetivada nas linhas 39 e 40.

Na linha 36 a função *get_queue_by_ip*() retorna a fila que o fluxo deverá ser colocado, baseado nos endereços IP de origem e de destino do fluxo. Quando não é identificada uma fila, em função dela não ter sido configurada previamente pelo usuário, a função retorna um valor nulo. Se o valor retornado não for nulo, o fluxo é configurado para a fila determinada (linha 38). Se o valor for nulo, *setQueue* não é chamado e o fluxo é colocado na fila de prioridades padrão. Nas linhas 39 e 40 o fluxo é configurado no comutador.

4.5 Casos de uso do *Ethanol*

Por meio da API proposta, a rede pode ser adaptada às necessidades de seus administradores, que podem utilizar informações de aplicações e sistemas internos para melhorar o desempenho da rede. Por ser uma interface livre e aberta, o *Ethanol* permite utilizar equipamentos de fabricantes diferentes, diminuindo o $CAPEX$ (*Capital Expenditure*) e reduzindo o efeito de aprisionamento tecnológico, ou seja, o usuário fica independente do fabricante de hardware. O uso de uma interface de programação aumenta o controle e o conhecimento do administrador de rede sobre o comportamento da sua rede. Desta forma, caso ocorra uma falha, o administrador pode analisar e alterar o código que está utilizando, sem depender de fornecedores. Nas subseções a seguir fazemos um paralelo dos desafios apresentados na seção 3.2 e algumas possíveis aplicações do modelo da figura 4.3. As aplicações propostas não são necessariamente inéditas, mas contribuem para mostrar a versatilidade da arquitetura.

4.5.1 Características variáveis dos enlaces

Por meio das classes *Radio* e *Station*, o controlador *Ethanol* é pode obter informações sobre as características dos enlaces com cada uma das estações conectadas à rede gerenciada pelo controlador. As condições da topologia local, por exemplo, podem ser obtidas utilizando $getChannelInfo()$ e $getNoiseInfo()$. Estas informações podem ser derivadas do próprio funcionamento da relação controlador-roteador *Ethanol*, onde o roteador sempre troca informações com o controlador. Este último é capaz de montar um grafo contendo os pontos de acesso. Este grafo associado aos eventos relativos ao processo de associação permitem tomar decisões para acrescentar (ou remover) as estações que estão conectadas a cada ponto de acesso. Este processo permite manter um grafo atualizado da rede.

O acesso a diversos contadores das estações, mediante utilização das mensagens de "*WLAN Radio Measurements*" definidas em IEEE 802.11k, permite ao controlador obter uma visão global da qualidade dos enlaces e do ambiente em torno de cada ponto de acesso sem fio. Assim, o controlador *Ethanol* permite gerenciar a qualidade dos enlaces ajustando a taxa de transmissão, largura do canal e potência de transmissão dos pontos de acesso que compõem a rede.

4.5.2 Mobilidade

O controle da mobilidade dos usuários pode ser melhorado utilizando o *Ethanol*. Este controle pode ser implementado com suporte de mensagens de localização ($getLocation()$) e da identificação dos pontos de acesso ao alcance de uma estação sem fio ($getAPsInRange()$). A utilização conjunta de protocolos de transição rápida, como IEEE 802.11r e IEEE 802.11v, auxiliam na redução do impacto da mobilidade sobre a experiência dos usuários. Ao mesmo tempo, melhor conhecimento pelo controlador da carga de usuários sobre um ponto de acesso ($getNumClients()$) e do seu ambiente de RF ($getNoiseInfo()$, $getChannelInfo()$ e $getLinkMeasurement()$) permite a ele chegar a uma decisão que melhor atinja aos requisitos globais da rede. Assim os clientes podem ser associados ao AP com o melhor sinal ou com melhores características, como número de clientes, quantidade de banda utilizada no momento ou menor quantidade de interferência.

4.5.3 Qualidade de serviço

A qualidade de serviço pode ser melhor gerenciada a partir do controlador *Ethanol*. Utilizando um controlador *Ethanol* é possível definir as prioridades de fluxos tanto

na rede cabeada quanto na rede sem fio. O *Ethanol*, ao implementar o protocolo OpenFlow e estendê-lo, pode alterar as filas de prioridade dos portos da rede Ethernet e encaminhar os quadros de acordo com a configuração destas filas. Ao mesmo tempo, o *Ethanol* tem acesso aos recursos disponíveis pelo protocolo IEEE 802.11e para a rede sem fio, o que o permite alterar os parâmetros de contenção para cada classe de tráfego. A capacidade do controlador de alterar os quadros transmitidos na rede permite um controle mais fino destes fluxos na comunicação entre a rede sem fio e a rede Ethernet.

4.5.4 Segurança

A arquitetura proposta não acrescenta recursos de segurança na transmissão dos dados, deixando este trabalho para o protocolo IEEE 802.11i (incorporado à IEEE 802.11-2007). Contudo a abordagem SDN proposta permite outras melhorias.

A existência de um controlador sensível às transmissões sem fio permite que quadros de autenticação IEEE 802.11x possam ser encaminhados diretamente pelo controlador ao servidor de autenticação RADIUS. Contudo, ao capturamos os eventos de autenticação em $evUserAuthenticating()$, o controlador pode responder diretamente ao ponto de acesso no caso de uma reautenticação, acelerando o processo de *handoff* para usuários móveis.

A forma de solicitação de uma autenticação também pode ser alterada, sem afetar as estações. No ambiente de rede local sem fio, as estações solicitam autenticação de rede utilizando o protocolo IEEE 802.1x ou via uma frase senha (*passphrase*) como definido em [IEEE802.11, 2012, p.1163][2]. Suponhamos agora o seguinte cenário: uma estação deseja conexão à rede sem fio em uma empresa parceira ou na rede de um outro provedor de acesso. A estação continua fazendo a solicitação via IEEE 802.1x, contudo quando a solicitação chega ao ponto de acesso sem fio, o evento $evUserAuthenticating()$ é chamado, sendo tratado pelo controlador. O controlador, identificando a solicitação, pode retransmiti-la para o servidor de autenticação adequado, inclusive por outro método (não é necessário que seja IEEE 802.1x). A estação pode ser autorizada ao receber a confirmação do servidor de autenticação. Simultaneamente, o controlador pode tomar outras providências, tais como isolar o tráfego desta estação mediante a criação de uma rede virtual para ela. Desta forma, um usuário móvel em um aeroporto poderia conectar a uma rede Wi-Fi de um provedor público utilizando seu usuário e senha corporativos. Os dados de autenticação seriam retransmitidos pelo controlador SDWN do provedor para um servidor de autenticação

[2]Não estamos considerando aqui as opções de sistemas abertos ou obsoletos, também denominados Pre-RSNA por [IEEE802.11, 2012, sec.11.2].

na empresa do usuário. Este servidor de autenticação faria a autorização e o provedor iria cobrar diretamente da empresa o uso da rede feito pelo usuário no aeroporto.

O controlador possui ainda a capacidade de monitorar todo o ambiente de rede. Utilizando métodos como os propostos em Opensketch (Yu et al. [2013]) e Jigsaw (Cheng et al. [2006]) e aproveitando das funcionalidades da API do *Ethanol*, podemos obter informações do ambiente de RF e de cada enlace mantido com as estações, bem como solicitar informações às estações, usando os recursos do protocolo IEEE 802.11k. Desta forma podemos, por exemplo, identificar pontos de acesso não autorizados, funcionando dentro da área de cobertura da rede sem fio.

4.5.5 Localização de usuários

Um controlador *Ethanol* pode localizar uma estação sem fio. O controlador, aproveitando o recurso fornecido pelo protocolo IEEE 802.11v, pode utilizar o método $getLocation()$ para obter a localização de estações sem fio que possuam este recurso disponível. Assim a estação irá fornecer os dados de GPS disponíveis.

Mesmo que este recurso não esteja disponível, ainda existe a possibilidade de utilizar métodos indiretos. Por exemplo, usando $getAPsInRange()$ fornecido pela entidade *Station*, podemos obter os dados dos pontos de acesso que estão ao alcance de uma estação. A partir desta informação, associada às informações fornecidas por $getLinkMeasurement()$, é possível obter a intensidade do sinal e obter a distância aproximada usando o conceito de FSPL (*Free Space Loss*). Com isto, sabendo a localização dos pontos de acesso controlados, é possível, mediante trilateração, obter a região aproximada da localização da estação.

4.5.6 Configuração

O *Ethanol* permite a configuração dos pontos de acesso. A arquitetura foi proposta considerando acesso de leitura e escrita aos parâmetros de configuração de redes sem fio IEEE 802.11. Desta forma o controlador pode efetuar tarefas como alocação dinâmica de canais entre os diversos pontos de acesso que controla, para minimizar a interferência.

Algoritmos sofisticados de monitoramento e controle podem também ser criados a partir da arquitetura proposta. Por exemplo, o tráfego de rede pode ser verificado a cada fluxo estabelecido e o padrão de fluxos vindos de uma estação sem fio pode ser avaliado. Podemos identificar, desta forma, que uma estação está infectada com um software malicioso que está tentando realizar, por exemplo, um ataque de negação de serviço. O controlador pode então desconectar esta estação da rede sem fio ou colocá-la

de quarentena, impedindo que ela tenha acesso à rede, durante um período ou até que uma ação do administrador de rede seja informada ao controlador.

A possibilidade de conhecer como é o tráfego da rede permite, por exemplo, que um operador de um provedor de banda larga tenha informações para identificar um problema em uma residência, tanto no enlace com fio quanto no enlace sem fio, mesmo que o usuário doméstico não tenha conhecimento técnico para verbalizar corretamente o que está acontecendo.

4.6 Conclusão

Vimos que a arquitetura proposta apresenta muitas potencialidades de uso, indo além do oferecido pelos controladores proprietários, ao mesmo tempo que aproveita melhor os recursos fornecidos pelos protocolos de gerenciamento mostrados na seção 3.1.2. No próximo capítulo apresentamos o protótipo desta arquitetura. Com este protótipo exploramos alguns dos cenários mostrados neste capítulo, avaliando seu funcionamento e desempenho em quatro estudos de caso.

Capítulo 5

Avaliação experimental do protótipo

Para testarmos a arquitetura proposta no capítulo 4, construímos um protótipo que implementa uma parte do modelo apresentado. Utilizando este protótipo, construímos alguns estudos de caso que nos permitiram avaliar a viabilidade da aplicação da abordagem SDWN proposta.

Montamos quatro estudos de caso: (a) implementamos um controle do fluxo de transmissões de quadros ARP entre as redes Ethernet e as redes locais sem fio IEEE 802.11, descrito na seção 5.1; (b) realizamos um experimento de controle de fluxo de pacotes com controle de filas de prioridade, considerando tráfego de estações Ethernet e sem fio direcionados a um servidor, conforme mostrado na seção 5.2; (c) implementamos um controle de associação a um conjunto de pontos de acesso sem fio, autorizando a conexão baseado na carga em cada ponto de acesso, como apresentado na seção 5.3; e (d) apresentamos um método de detecção de falhas na interface de rede sem fio na seção 5.4.

Em seguida, nas próximas seções deste capítulo, apresentamos os quatro estudos de caso. Na última seção deste capítulo apresentamos uma avaliação do uso de processador e memória no controlador e no roteador *Ethanol*.

5.1 Controle de transmissões ARP

O protocolo IP na versão 4 utiliza uma resolução dinâmica de endereços de hardware obtida pela implementação o protocolo ARP (*Address Resolution Protocol*), cujo funcionamento está esquematizado na figura 5.1.

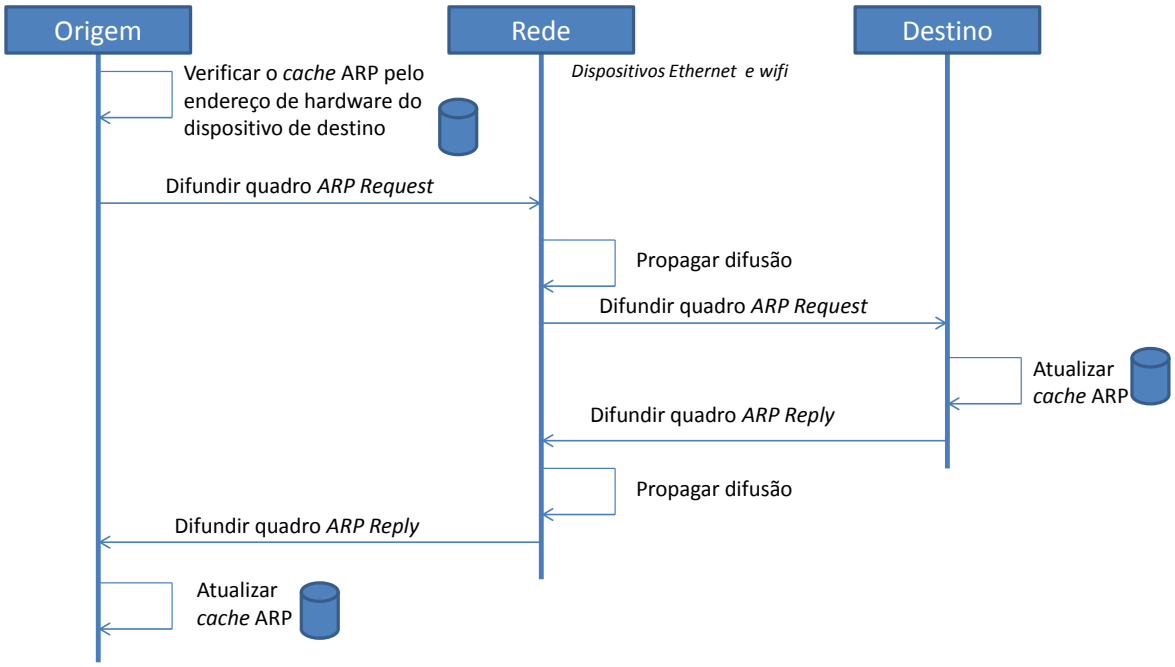


Figura 5.1: Processo de transação para o protocolo de resolução de endereço

A operação do protocolo ARP envolve codificar o endereço IP do hospedeiro de destino (o dispositivo do qual se deseja conhecer o endereço de hardware) em uma mensagem de difusão ("*broadcast*"). Esta mensagem é enviada para toda a rede local, desta forma alcançando o hospedeiro de destino, que responde ao hospedeiro de origem (estação que enviou a mensagem) qual é seu endereço de camada de enlace. Isso é feito usando um método de solicitação/resposta. Um formato especial de mensagem é usado para as mensagens ARP, que são encaminhadas para a camada local de enlace de dados para a transmissão.

Cheng et al. [2006] concluíram que em seus experimentos os pacotes ARP transmitidos na rede local podem consumir quase 10% do tempo de comunicação nas redes sem fio monitoradas. Este tráfego ARP é transmitido via difusão e desta forma todas as transmissões de ARP da rede com fio também são enviadas em difusão para o canal de rede sem fio, consumindo tempo útil de transmissão. Além disso, este tráfego escala com a quantidade de usuários na rede, enquanto que a capacidade de transmissão da rede sem fio permanece constante.

Podemos usar uma abordagem SDN para reduzir o impacto destas transmissões[1]. Uma vez que o controlador tenha conhecimento do mapeamento entre os endereços IP e MAC e do mapeamento entre o endereço MAC e o porto do comutador, o controlador

[1]Controladoras comercias fornecem um serviço semelhante contudo com menor abrangência que nossa proposta. A Cisco System fornece um recurso de *proxy ARP*. Contudo este recurso não funciona com clientes passivos, isto é, aqueles que tenham endereço IP estático [Cisco Systems, 2010, p.7-71].

pode resolver os pedidos ARP utilizando o **Algoritmo 1**. Assim, filtrando o tráfego ARP encaminhado em todos os pontos de acesso *Ethanol*, o tempo disponível para transmissão de dados dos usuários é aumentado. Note que podemos transformar uma mensagem *ARP Request*, que é uma transmissão de difusão, em uma resposta direta à estação que solicita esta informação, utilizando na tabela de mapeamento existente no controlador. Também podemos transformar uma mensagem *ARP Response*, outra difusão na rede, em uma resposta direta à estação que solicitou a informação.

Algoritmo 1 Controle de transmissão de ARP na rede

```
1: arpTable ← ∅                          ▷ tabela hash mapeando um endereço de hardware em um endereço IP
2: macToPort ← ∅ ▷ mapeia um endereço de hardware para um porto do comutador
3: function evPacketIn(event)                          ▷ event é um parâmetro do POX
4:     packet ← event.parsed
5:     mac_dst ← packet.mac_dst
6:     IP_dst ← packet.IP_dst
7:     mac_src ← packet.mac_src
8:     IP_src ← packet.IP_src
9:     macToPort[mac_src] ← event.port          ▷ mantem o mapeamento macToPort
10:    arpTable[mac_src] ← packet.IP_src                    ▷ atualiza a tabela ARP
11:    if packet.type! = ARP then ▷ quadros não ARP são transmitidos diretamente
12:        (new Flow(event)).apply()
13:    else                                                 ▷ trata todos os quadros ARP
14:        if !findIP(arpTable, IP_dst) then
15:            flood(packet)                ▷ destinatário desconhecido: inunda a rede
16:        else                                             ▷ destinatário conhecido:
17:            if packet.arp = REQUEST then ▷ (a) responde diretamente a origem
18:                SendArpReply(mac_dst, mac_src, macToPort[mac_src])
19:            else                ▷ (b) transmite resposta diretamente ao destinatário
20:                SendArpReply(mac_src, mac_dst, macToPort[mac_dst])
21:            end if
22:        end if
23:    end if
24: end function
```

Com a execução deste algoritmo, para a estação de origem o processo é o mesmo do funcionamento do protocolo original, mostrado no lado esquerdo da figura 5.1. Portanto, a estação consulta sua tabela local e, ao identificar que não possui o endereço de hardware do destino, gera uma mensagem *ARP Request* que envia para a rede. Contudo a nossa rede é composta de dispositivos SDN que ao receberem o quadro, geram um evento de "*Packet In*". Este evento gera uma mensagem para o controlador. O controlador verifica se possui a informação. Podem ocorrer dois processos, caso

conheça ou não a informação em sua tabela ARP.

Se o controlador desconhece o mapeamento IP x MAC, o controlador programa uma difusão na rede e, assim, a mensagem *ARP Request* é encaminhada para todos os portos do dispositivo. A rede, neste caso, comporta-se como no protocolo ARP original. Contudo, o controlador aproveita para cadastrar as informações da estação de origem: porto que está conectado, o endereços de hardware e o endereço IPv4.

Eventualmente a mensagem chega ao destino, que a processa normalmente: identifica a mensagem de solicitação, gera uma mensagem de resposta denominada *ARP Reply* e a encaminha para a rede sob a forma de difusão. Desta forma para a estação de destino, o funcionamento do protocolo ARP também parece inalterado. O quadro de resposta da estação é enviado para o comutador, um dispositivo SDN, e, assim, um novo evento "*Packet In*" é gerado. O controlador identifica que é uma mensagem *ARP Reply* para a estação de origem. Porém, desta vez, é possível responder diretamente a esta estação de origem (que está procurando o endereço de hardware), pois as informações desta estação foram armazenadas durante a etapa do *ARP Request*. O controlador agora programa um fluxo diretamente para a estação de destino. O quadro enviado é ainda um *ARP Reply*. Esta mensagem aparece para a estação de origem como uma mensagem de difusão, porém ela foi encaminhada somente para o porto onde a estação está conectada, funcionando, sob o ponto de vista da rede, como se fosse uma mensagem de *unicast*.

Caso o controlador já conheça o endereço de hardware do destino, o processo é simplificado. Neste caso a estação de destino não é contactada. O próprio controlador, na fase do *ARP Request* da figura 5.1, gera a mensagem de resposta para a estação de origem, descartando a mensagem de *ARP Request*.

Esta lógica é apresentada no **Algoritmo 1**. Quando o controlador não reconhece o endereço IP, ele inunda o pedido em todos os portos, como mostrado na linha 15. Se o controlador já conhece o endereço IP, ele responde a solicitação diretamente para a origem, como vemos na linha 18. Quando recebe o pacote de resposta ARP, o controlador responde diretamente a estação de origem, que é identificada como destino da mensagem de resposta na linha 20. O controlador gera, nestes dois casos, uma mensagem ARP, portanto uma mensagem de difusão, contudo ao observarmos o comportamento da rede, o resultado equivale à transmissão de uma mensagem *unicast* para o hospedeiro solicitante. Todos os outros tipos de tráfego são transmitidos como em um comutador de camada 2 (linha 12).

5.1.1 Configuração do experimento

Montamos o experimento esquematizado na figura 5.2 para testar o controle de fluxos feito com o **Algoritmo 1**. Neste estudo de caso, temos um roteador *Ethanol* conectado ao controlador *Ethanol* por um canal seguro Ethernet. Ao roteador conectamos um cliente sem fio executando um analisador de pacotes (*tcpdump*[2]) que captura todos os quadros que chegam à sua interface sem fio. Ao roteador também ligamos, pela rede Ethernet, um conjunto de duas ou mais estações.

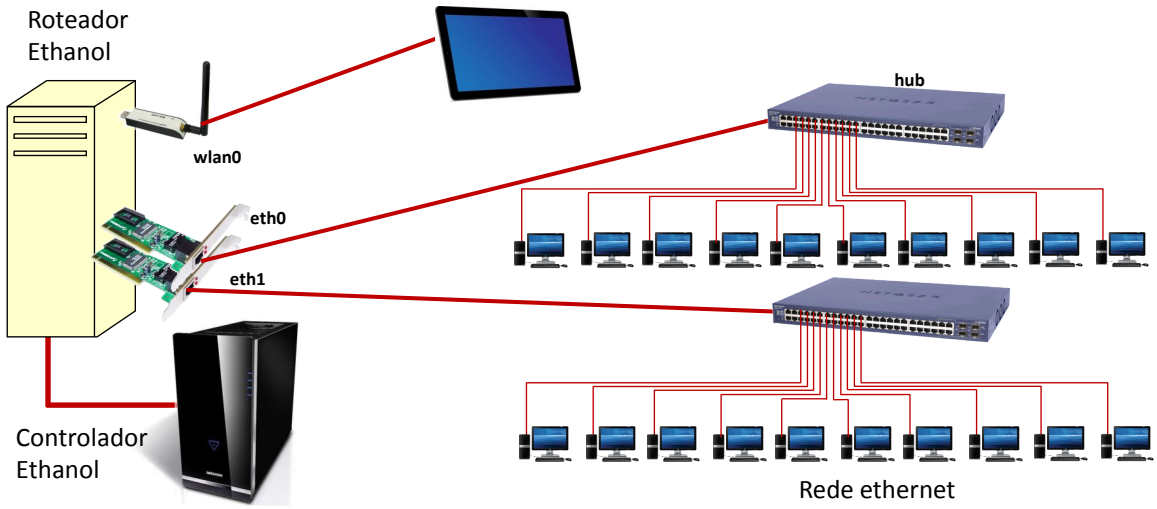


Figura 5.2: Esquema de conexão para experimento de controle de fluxo ARP

Os clientes Ethernet geram tráfego para outros clientes Ethernet e também para o cliente sem fio, utilizando *ping*. Os quadros, que chegam ao cliente sem fio, são identificados e capturados pelo *tcpdump*. Depois de um período de captura, os dados são analisados para verificarmos como o tráfego se comportou com o controlador gerenciando o tráfego ARP ou funcionado como um comutador de camada 2 tradicional.

O tráfego é gerando em cada estação da rede Ethernet com destino ao um subconjunto formado por cerca de $\frac{1}{5}$ do total da rede Ethernet e a estação sem fio. A cada solicitação de *Ping*, a *tabela ARP* local é limpa, forçando a geração de novas requisições ARP. Como sugestão para trabalhos futuros indicamos o levantamento e utilização de *traces* reais de redes para observação do comportamento do nosso algoritmo, bem como para estudo comparativo com o caso apresentado por Cheng et al. [2006].

[2] `http://www.tcpdump.org/`

5.1.2 Resultados

O fluxo de dados percebido pelo cliente sem fio durante o experimento é mostrado na figura 5.3. Este gráfico foi construído com base nos quadros capturados pelo *tcpdump* na interface sem fio do cliente. O experimento foi executado durante 600 segundos, havendo uma alternância entre o controle do tráfego ARP e o funcionamento do controlador como um comutador de camada 2 tradicional. Os intervalos foram, como pode ser visto na figura, de 150 segundos (como indicado pelas sequências de faixas brancas e cinzas).

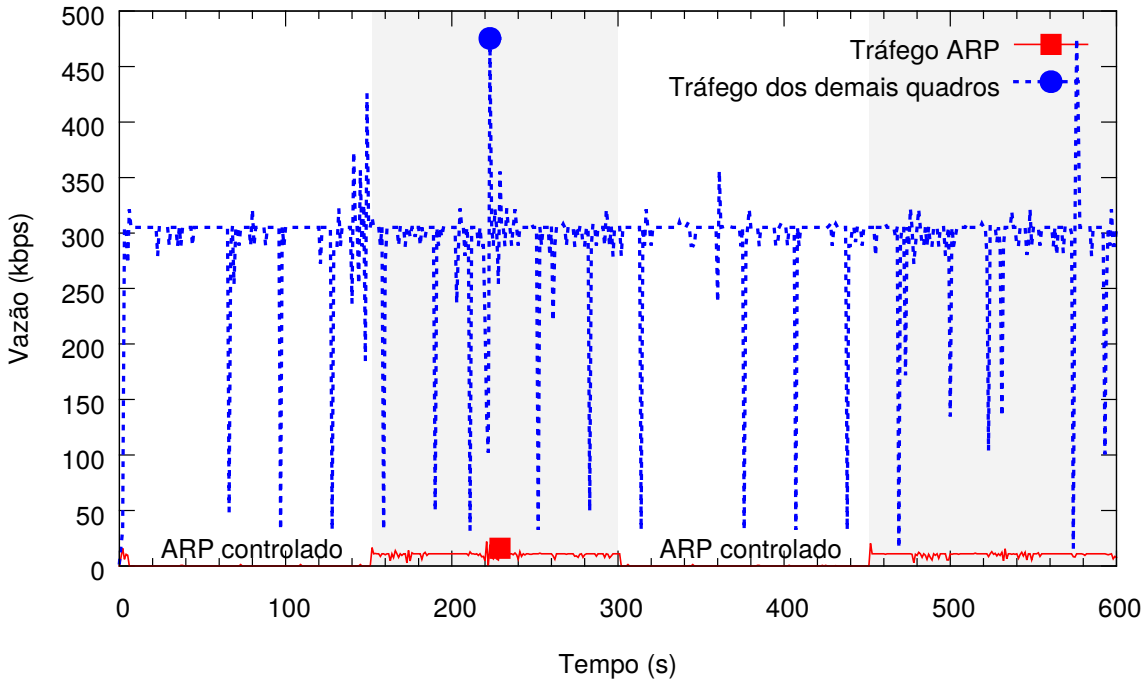


Figura 5.3: Vazão no cenário de controle de fluxo ARP

O experimento começa com o controle do tráfego ARP ativado. Notamos que no início do experimento existem diversas solicitações ARP que são transmitidas na rede como difusão, pois o controlador ainda não conhece o mapeamento endereço IP x endereço de hardware necessário para simplificar as comunicações. Desta forma, vemos na parte inferior esquerda do gráfico um pequeno pico, correspondendo a este tráfego. Como este tráfego é de difusão, ele chega, portanto, ao cliente sem fio.

Após esta primeira etapa, quando o controlador já possui conhecimento do mapeamento dos endereços de hardware, o tráfego ARP percebido pela estação sem fio fica baixo, chegando a quase zero. Analisando cada quadro ARP que efetivamente chega ao cliente sem fio neste período, identificamos que os picos correspondem às solicitações da estação sem fio e às respostas para esta estação.

Ação	**Controlador com função ARP (Número de pacotes)**	
	Ativada	**Desativada**
Transmitidos por **Difusão**	37	19.720
Enviados na **Requisição** ARP	9744	-
Enviados na **Resposta** ARP	22	-
Total	9.803	19.720

Tabela 5.1: Processamento de pacotes ARP pelo controlador em número de pacotes

Na tabela 5.1 apresentamos o processamento de quadros ARP pelo controlador, indicando o número de quadros ARP processados durante cada uma das fases do experimento. Esta tabela apresenta as seguintes informações: o número total de quadros processados ("Total"), o número total de quadros que são encaminhados por difusão ("Difusão"), o número total de quadros que são encaminhados pelo controlador a partir de uma requisição de uma estação ("Requisição") e o número total de quadros que são encaminhados pelo controlador a partir de uma resposta de uma estação a um quadro de ARP Request ("Resposta"). Apresentamos duas colunas para cada valor. A coluna da esquerda apresenta o valor da informação quando o controlador ARP está ativado e a coluna da direita mostra o valor quando o controlador está desativado (funcionando somente como um comutador de camada 2 tradicional).

O número total de quadros processados pelo controlador quando o controle do fluxo ARP está desativado é praticamente o dobro do valor observado quando o controlador está ativado. O aumento da quantidade de quadros que chegam ao controlador, na fase que o controle está desativado, ocorre porque para cada *ARP Request* existe um *ARP Reply* que inunda a rede. O valor observado na linha "Total" para o controlador desativado não é exatamente o dobro da quantidade observada para o controlador ativo. Isto ocorre porque, com o controle ativado, ainda existem algumas transmissões de *ARP Reply*, desta forma são gerados dois quadros no processo de resolução do endereço de hardware, mesmo com o controlador ativo.

Com o controle desativado, todo o tráfego observado é de inundação (difusão), como mostrado na coluna da direita da tabela 5.1. Como vemos na tabela este valor corresponde a 100% dos quadros que trafegam na rede. Este é o comportamento esperado de uma rede com o protocolo ARP para IP versão 4.

Com o controlador executado o controle sobre o fluxo ARP, as inundações chegam a quase zero. As inundações ainda são necessárias para que o controlador conheça a rede e monte a sua tabela de mapeamento. Quando o mapeamento está feito, o controlador responde diretamente às requisições, representada pela coluna da esquerda em "Requisição" na tabela 5.1.

O controlador ainda gera um pequeno número de respostas para *ARP Reply*.

Estas ocorrem no início do processo de conhecimento da rede, quando o controlador desconhece o endereço de hardware do destino e tem que fazer uma inundação, mas ao receber a resposta do hospedeiro de destino, o controlador já é capaz de enviar a resposta diretamente para o hospedeiro de origem. Este processo ocorre também quando uma nova estação entra na rede, por exemplo, quando um novo computador é conectado à rede e o endereço de hardware deste computador não é conhecido pelo controlador. Não fizemos este tipo de simulação no nosso experimento.

O número de difusões detectadas pela estação é sempre menor ou igual ao número de estações na rede, quando o controlador está com a função de controle ARP ativada. Contudo não existe uma relação direta entre este dois parâmetros, pois todos os pacotes de *ARP Request* geram informação útil para o controlador. Por exemplo, em uma rede com 40 estações podemos ter dois casos extremos. No primeiro caso, somente uma estação gera *ARP Request* para as demais, neste caso teríamos uma situação onde os valores serão 39 para "Difusão" e 39 para "Resposta". No segundo caso, metade das estações envia um *ARP Request* para exatamente a outra matade da rede. Neste caso, os valores serão 20 para "Difusão" e 20 para "Resposta". Desta forma vemos que a ordem e os destinos das solicitações influenciam os valores de 'Difusão" e "Resposta" obtidos.

Experimento	**Tempo de execução** (em μs)	**Intervalo de confiança** $1 - \alpha = 95\%$	
		Mínimo	**Máximo**
Controle ARP desativado	1082,87	1076,82	1088,93
Controle ARP ativado	1135,61	1132,62	1138,60

Tabela 5.2: Tempo de processamento de pacotes ARP pelo controlador (em microsegundos)

A tabela 5.2 mostra o tempo de execução do procedimento *evPacketIn* quando o controlador realiza o controle do tráfego ARP e quando não está controlando o fluxo do ARP. Estes valores são mostrados em duas linhas - **Controle ARP desativado** e **Controle ARP ativado**. Vemos que existe um acréscimo de menos de 5% no tempo de execução. Portanto, o processamento adicional dos pacotes ARP pelo controlador não impõe uma carga muito severa ao controlador, enquanto os benefícios para a rede sem fio são bastante significativos.

A figura 5.4 mostra a distribuição cumulativa dos valores de RTT, medidos a partir da estação de rede sem fio. Utilizamos o RTT (*Round Trip Time*) para verificar o atraso inserido pelo *Ethanol* na propagação de um quadro ARP. Esta grandeza mede o tempo gasto para o envio da informação do fluxo ao controlador, em função do evento "*Packet In*", mais o tempo de processamento no controlador e o tempo do registro do fluxo no comutador pelo controlador. Neste gráfico medimos o RTT para

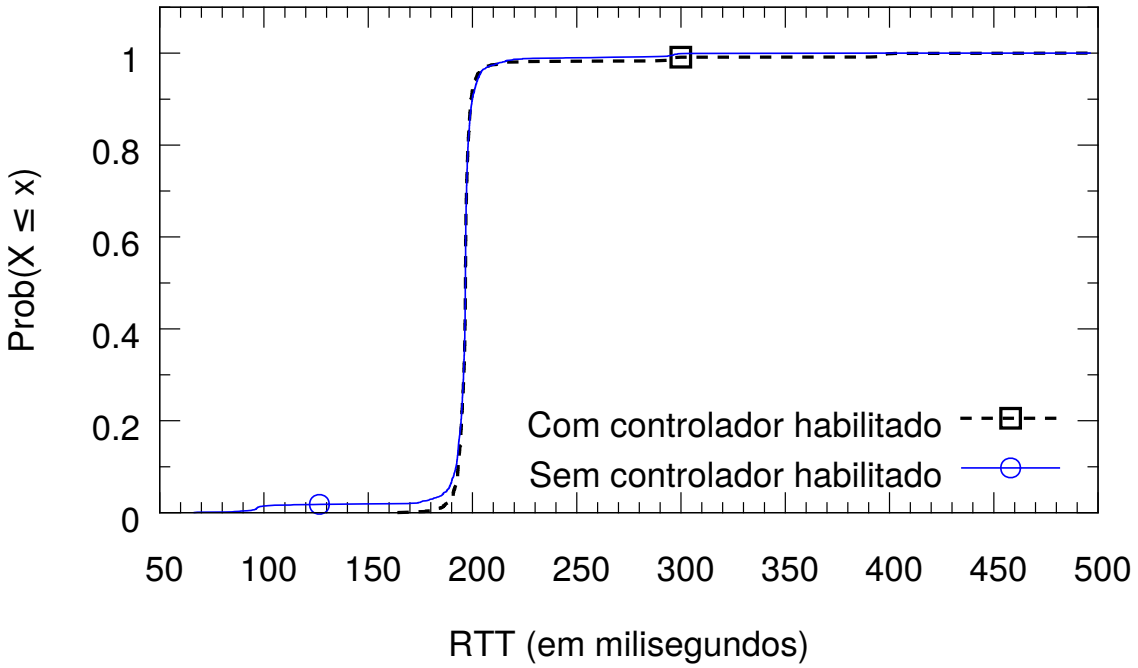


Figura 5.4: Distribuição cumulativa de frequências dos RTT

uma solicitação feita com o programa *Ping*, com a tabela ARP da estação limpa. Desta forma, cada solicitação do *Ping* irá gerar uma solicitação para resolução de ARP. Com o controle ARP ativado, temos 95% dos valores para RTT menores ou iguais a 202 milissegundos. Com o controle ARP desativado, temos 95% dos valores para RTT menores ou iguais a 203 milissegundos. Desta forma o uso do nosso algoritmo não afeta significativamente o tempo de resposta para o primeiro pacote transmitido. O atraso representado pela propagação do quadro na rede sem fio se sobrepõe ao atraso gerado pela mensagem ao controlador.

5.2 Implementação de QoS para tráfego Ethernet e sem fio

O *Ethanol* permite realizar o controle de tráfego nas redes Ethernet e sem fio. Desta forma podemos influenciar favoravelmente um fluxo em relação a outro. Utilizando o **Algoritmo 2** podemos controlar a vazão dos fluxos de dados baseado na identificação do endereço IP de origem. Para tanto, é necessário possuir uma tabela que mapeia $IP\ de\ origem \Rightarrow Fila\ de\ prioridade$. A geração deste mapeamento não está indicada no algoritmo e pode ser, por exemplo, um mapeamento estático configurado pelo administrador de rede ou um mapeamento dinâmico de alto nível que compara os perfis

de tráfego com políticas de alto nível definidas pelo administrador de rede. No nosso experimento, este mapeamento é estático e está contido em um arquivo de configuração que é lido na carga do programa.

Algoritmo 2 Controle de fluxo de dados utilizando fila de prioridade

```
1: function evPacketIn(event)                    ▷ event é um parâmetro do POX
2:     packet ← event.parsed
3:     new_queue ← getMapedQueueByIP(packet.next.srcip)   ▷ retorna a fila de
   encaminhamento em função do endereço IP de origem
4:     flow ← newFlow(event)  ▷ cria um novo fluxo na tabela de encaminhamento
   do roteador baseado nos parâmetros do quadro recebido pelo controlador
5:     flow.setQueue(new_queue)                  ▷ coloca o fluxo na fila correta
6:     flow.apply()
7: end function
```

O agente *Ethanol* gera comandos compatíveis com o *Openvswitch* para (1) criar e configurar as filas ou (2) para limpar as filas. Desta forma, mediante envio de mensagem do controlador para o agente *Ethanol* no roteador, as filas no roteador são manipuladas. Para cada classe podemos atribuir um valor de fluxo máximo e mínimo, mediante a criação de filas apropriadas no roteador *Ethanol*, utilizando os recursos fornecidos pela classe *AccessPoint*. Neste experimento, as filas são configuradas no roteador durante o processo de inicialização do controlador. Desta forma, no **Algoritmo 2** temos somente que direcionar um fluxo a uma fila pré configurada baseado em um critério (o endereço IP de origem).

A configuração de filas pode ser feita, também, utilizando mensagens para *netlink socket* (`http://www.linuxfoundation.org/collaborate/workgroups/networking/netlink`). A utilização do *netlink* permite realizar uma chamada, em espaço de usuário, para o *kernel* do sistema, acelerando o processo de configuração. Além disto, o uso do *netlink* evita a necessidade de configuração manual do caminho para os comandos do *Openvswitch*, forma utilizada atualmente no *Ethanol*. Contudo em função do tempo disponível para desenvolvimento durante a dissertação, descartamos a opção de usar o *netlink*. O desenvolvimento usando mensagens do *netlink* fica como uma sugestão para um aprimoramento do nosso protótipo.

Utilizamos o escalonador de fila padrão no linux, o HTB (Devera [2002]), que é utilizado para garantir a prioridade de fluxo, de modo que as taxas de transferências dos fluxos sejam proporcionais às dotações atribuídas. A seleção de qual escalonador será utilizado pode ser feita mediante mensagens *netlink*, que deixamos para trabalhos futuros.

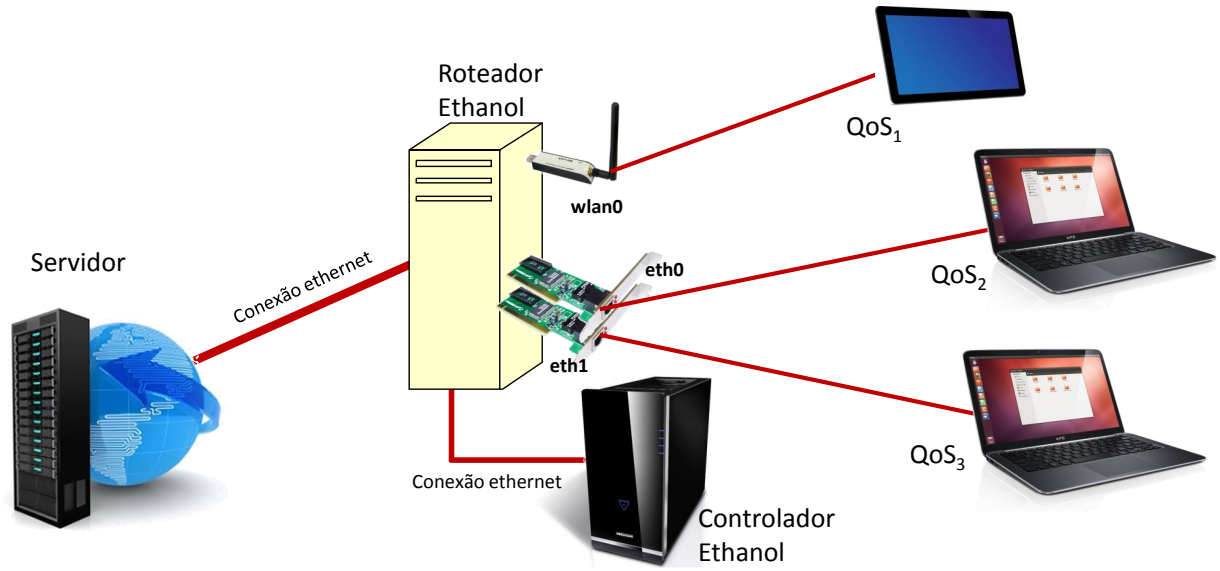


Figura 5.5: Esquema de conexão para experimento de controle de fluxo

5.2.1 Configuração do experimento

A figura 5.5 apresenta a montagem do experimento para teste do controle de fluxo usando filas de prioridade feito com o **Algoritmo 2**. O roteador está conectado a dois clientes via rede Ethernet e a um cliente sem fio, que está associado à sua BSS. Esta configuração emula uma rede corporativa, onde clientes sem fio e com fio desejam acesso a um recurso da Intranet.

Para geração do tráfego entre as estações e o servidor, utilizamos uma versão do software *BWPing*, desenvolvido no DCC/UFMG. Esta é uma ferramenta para medir os tempos de resposta entre dois hospedeiros e também a largura de banda, empregando UDP em IPv4.

Configuramos três filas no roteador *Ethanol*, com as seguintes proporções: $Q_1 = 6$, $Q_2 = 3$ e $Q_3 = 1$. Quando um novo fluxo é detectado no roteador, um evento "*Packet In*" é gerado no controlador. Este evento aciona o **Algoritmo 2**, que nos permite definir para qual fila será atribuído este novo tráfego. Quando o pacote é recebido, o controlador analisa sua tabela de atribuição de filas (que mapeia um endereço IP de origem para uma determinada fila de prioridade). Identificada a fila (ou selecionada a fila padrão caso não exista o mapeamento), o fluxo é atribuído à fila adequada. No enlace conectando o roteador com o servidor foi programado um limite máximo de vazão equivale ao máximo suportado pelo enlace sem fio.

O experimento consiste em duas fases: na primeira fase o cliente sem fio C_1 e um dos clientes Ethernet C_2 estão transmitindo. Já na segunda fase, o segundo cliente

Ethernet C_3 também passa a transmitir, de modo que as vazões são reajustadas para acomodar o novo tráfego, respeitando as atribuições de prioridade.

5.2.2 Resultados

A figura 5.6 mostra a vazão detectada pelo servidor e medida pelo *BWPing* a cada instante. O experimento teve uma duração total de 340 segundos. Nos primeiros 170 segundos, somente os clientes C_1 e C_2 transmitiam. Como as ativações entre os dois clientes não foram síncronas, observamos na parte mais à esquerda do gráfico um pico do cliente C_2. Notamos nos primeiros 170 segundos que a vazão conjunta fica abaixo de 800 kbps. Na segunda etapa do experimento, quando o cliente C_3 também transmite, as vazões de C_1 e C_2 são ajustadas para comportar o novo tráfego.

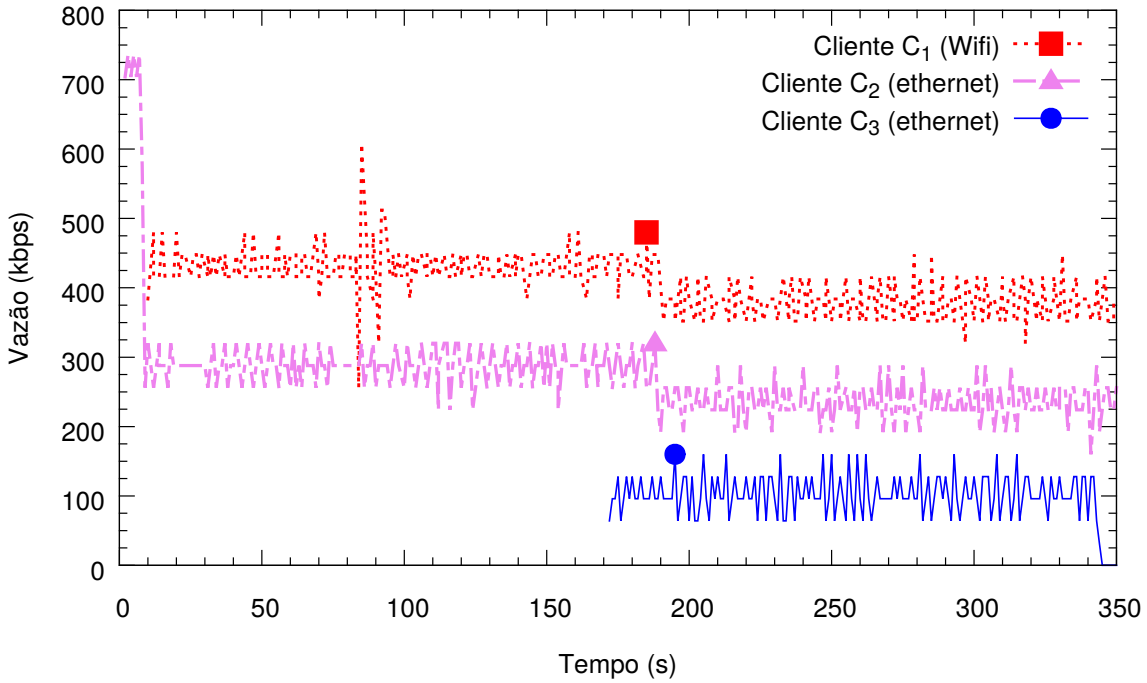


Figura 5.6: Tráfego gerado pelas 3 estações com filas de prioridade no roteador *Ethanol*

Realizamos ainda um experimento de controle, onde somente o cliente C_1, a estação sem fio, efetua transmissões na rede. Desta forma é possível verificar que a vazão máxima que o cliente C_1 consegue atingir é da ordem de 768 kbps. Este valor máximo corrobora as distribuições que observamos na figura 5.6, quando a banda era compartilhada por 2 ou 3 estações.

Realizamos ainda um outro experimento, mantidas as mesmas condições, porém sem o controle de fluxo ativado. Neste experimento utilizamos o comutador *forwarding.l2_learning* fornecido pela biblioteca do POX. Como observamos na figura 5.7,

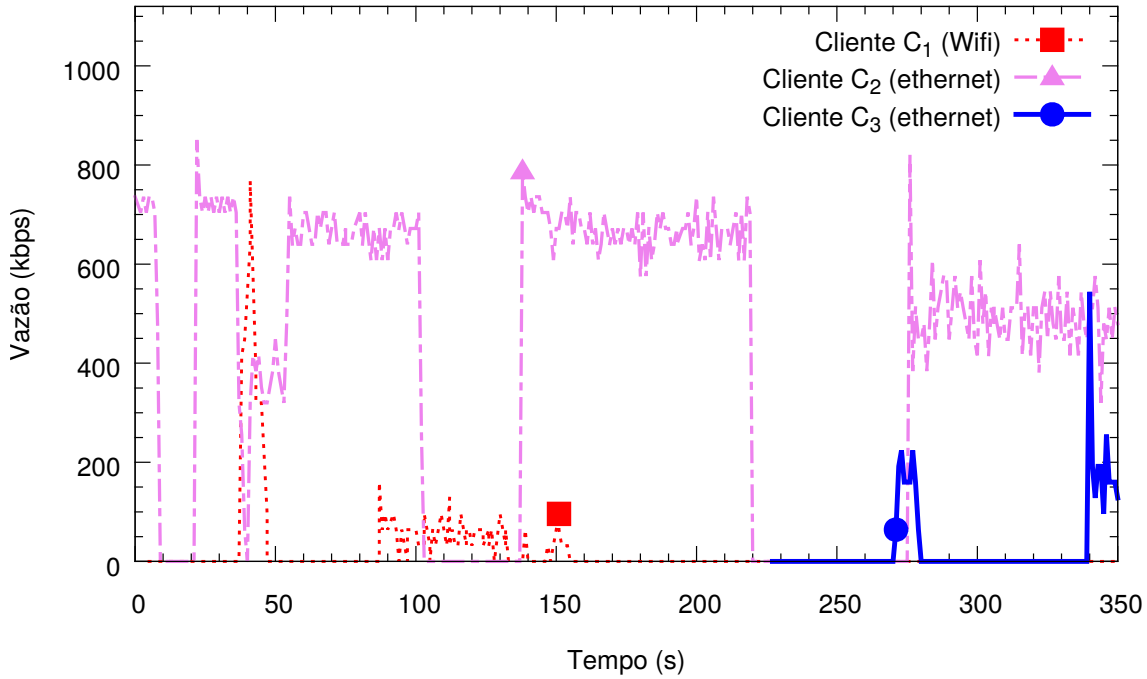


Figura 5.7: Tráfego gerado com as 3 estações no roteador *Ethanol* sem controle de QoS

sem a priorização do tráfego do cliente sem fio (C_1), este cliente passa a ter pouco acesso ao servidor. Na segunda etapa do experimento, o cliente C_1 nem mesmo conseguiu transmitir algum quadro que chegasse ao servidor.

Medida	**Tempo de execução**	**Intervalo de confiança** $1-\alpha = 95\%$	
		Mínimo	**Máximo**
Sem controle Qos	631,65	610,09	653,21
Com controle Qos	1015,83	964,18	1067,48

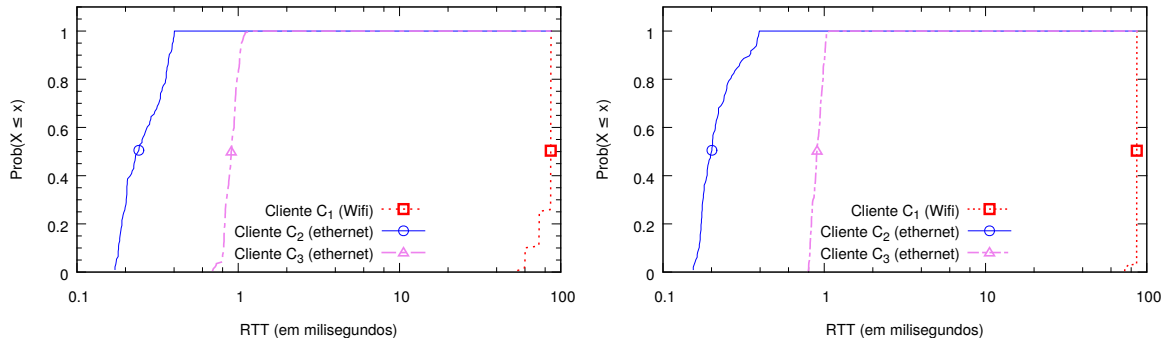
Tabela 5.3: Tempo de processamento de pacotes pelo controlador (em microsegundos)

A tabela 5.3 mostra o tempo de execução do procedimento *evPacketIn* executando o **Algoritmo 2** com os valores para o intervalo de confiança em cada uma das situações. Indicamos duas situações: quando o controlador está executando o controle de fluxos e aplica as regras de encaminhamento para as filas de prioridade, representado pela linha **Com controle de QoS** da tabela, e quando o controlador funciona somente como um comutador de camada 2, sem realizar o enfileiramento dos quadros, indicado pela linha denominada **Sem controle de QoS**. As medições de tempo consideram o intervalo entre tempo inicial quando o procedimento *PacketIn* começa a ser executado no controlador e o tempo final correspondente a última instrução antes de retornar. Vemos que existe um acréscimo da ordem de 61% no tempo de execução.

Contudo o tempo total de execução acrescenta menos de $0,5$ ms de atraso na transmissão dos pacotes. Não avaliamos nosso algoritmo em uma rede de alta velocidade. Como sugestão de trabalhos futuros, este algoritmo precisaria ser avaliado em redes Ethernet com velocidade de transmissão de 1 Gbps ou superior e em redes sem fio IEEE 802.11ac que atingem velocidades de 600 Mbps, para verificar se estes atrasos seriam significativos nestas redes. Também deveria ser avaliado utilizando outro controlador, como por exemplo o NOX, capaz de melhores desempenhos. O experimento também considerou somente o funcionamento reativo do controlador, ou seja, as regras para um fluxo somente são programadas no comutador quando este fluxo é detectado.

Na figura 5.8 mostramos a distribuição cumulativa dos valores de RTT, medidos a partir de cada uma das estações usadas neste experimento ao tentar acessar o servidor. Neste gráfico medimos o tempo necessário para uma solicitação feita com o programa *Ping* da estação para o servidor ser transmitida e recebida novamente pela estação. A escala do eixo das abscissas é logarítmica para facilitar a visualização da separação das curvas. Na figura 5.8a, mostramos a distribuição obtida para os valores do RTT para as estações C_1, C_2 e C_3 com o controlador realizando o controle de fluxos. Desta forma o quadro gerado pelo *ping* é colocado nas filas de prioridade definidas para cada cliente. Na figura 5.8b, mostramos a distribuição obtida para os valores do RTT para as estações C_1, C_2 e C_3 com o controlador funcionando como um comutador de camada 2 (que chamaremos de comutador normal). Este comutador normal foi implementado, sem modificações, utilizado o comutador *forwarding.l2_learning* fornecido pela biblioteca do POX.

Observamos para o cliente em fio, C_1, que o RTT máximo é 87 ms quando o controlador está ativado, e $86,8$ ms quando funciona como comutador normal, para 95% dos quadros transmitidos. Contudo para 24,5% dos quadros, o RTT cai para



(a) Medições realizadas com controlador ativado

(b) Medições realizadas com controlador desativado

Figura 5.8: Distribuição acumulativa de frequências para RTT

$73,7\ ms$ quando há controle de fluxos, porém cai somente para $86,6\ ms$ para um comutador normal. Isto mostra que há uma melhoria para uma parte do tráfego do cliente sem fio, em função da priorização de seus quadros pelo controlador. Para os clientes Ethernet, as distribuições são similares nas duas condições, apresentado pouca alteração. O valor do RTT máximo, para 95% dos quadros, é cerca de $0,4\ ms$ para o cliente C_2 em ambos os casos. Para o cliente C_3, o valor do RTT máximo para 95% do quadros é de $1,06\ ms$ com o controle de fluxo e de $1,03\ ms$ para o comutador normal.

5.3 Controle do processo de associação de estações sem fio

Durante o processo de associação de uma estação à rede sem fio, podemos controlar em qual dos pontos de acesso sem fio um cliente (estação) poderá se conectar. Desta forma é possível implementar um balanceamento de carga entre os pontos de acesso sem fio. Este controle pode ser feito utilizando o **Algoritmo 3** utilizando o evento $evUserConnecting$.

Estamos fazendo o balanceamento exclusivamente pelo número de clientes. Este critério utilizado é simplista. Em uma aplicação real não usaríamos somente este critério. Outros critérios adicionais deveriam ser utilizados. Por exemplo poderíamos considerar o nível de sinal, o tráfego total dos clientes atuais ou os valores SNIR (Signal to Noise plus Interference Ratio) em relação aos pontos de acesso. O critério utilizado no algoritmo foi selecionado como prova de conceito da viabilidade do controle da associação de estações sem fio.

Algoritmo 3 Controle do processo de associação de uma estação a um AP

```
1: function evUserConnecting(userMAC, apMAC)
2:     apsInRange ← AccessPoint.getApByMAC(apMAC).getApInRange()▷
       Retorna os pontos de acesso próximos ao AP que recebeu a solicitação
3:     minClients ← argmin_{ap ∈ apsInRange} {ap.numClients()}
4:     apClients ← AccessPoint.getApByMAC(apMAC).numClients()
5:     return apClients == minClients        ▷ retorna true para permitir a conexão
6: end function
```

No **Algoritmo 3**, inicialmente obtemos quais pontos de acesso estão ao alcance da estação que deseja conectar. Dessa forma sabemos quais são os potenciais pontos de conexão desta estação à rede sem fio. Obtemos o valor da menor quantidade de clientes ($minClients$) conectados a este conjunto de pontos de acesso. Caso a quantidade de clientes conectada ao ponto de acesso que está enviando a solicitação para o controlador

seja igual a *minClients*, autorizamos a conexão ao ponto de acesso. O processo de controle realizado pelo controlador *Ethanol* pode ser visto no diagrama 5.9. O processo normal de associação, realizado por um ponto de acesso IEEE 802.11 sem modificações, é constituído das etapas 1-5, 12 e 13. Note que sob a ótica do cliente (estação sem fio), o processo não foi alterado, pois as mensagens continuam as mesmas.

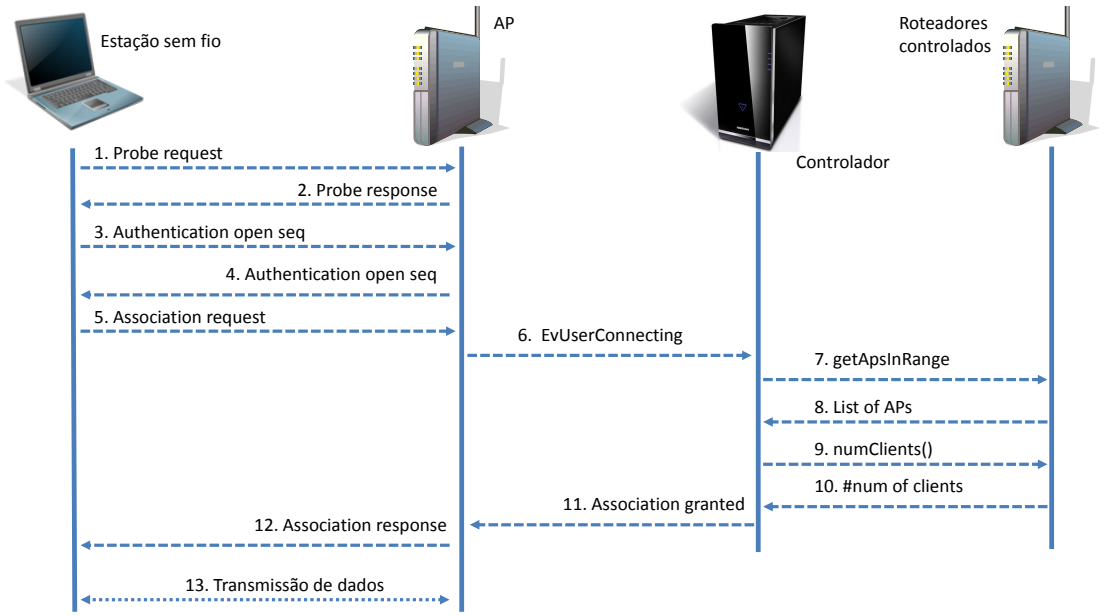


Figura 5.9: Controle do processo de associação de uma estação à rede sem fio realizado pelo *Ethanol*

No **Algoritmo 3** usamos uma aproximação para a lista de pontos de acesso ao alcance da estação que deseja conectar, em função da indisponibilidade do recurso oferecido pelo protocolo IEEE 802.11k no nosso equipamento. A melhor opção seria que a estação fornecesse esta informação, via mensagem IEEE 802.11k. Contudo nos nossos equipamentos, o protocolo IEEE 802.11k não está implementado, impedindo que o controlador *Ethanol* possa solicitar esta informação da estação. Desta forma, como os roteadores *Ethanol* realizam um monitoramento dos sinais de *beacon* a intervalos regulares, podemos identificar os pontos de acesso que estão próximos de cada roteador *Ethanol*.

5.3.1 Configuração do experimento

Para testar o algoritmo de controle de associação de estações sem fio baseado na carga, representada pelo número de estações conectadas ao ponto de acesso, montamos o experimento esquematizado na figura 5.10. O experimento consiste em dois pontos de acesso conectados via rede Ethernet ao controlador *Ethanol*, fornecendo a rede local

sem fio com o mesmo ESSID. Possuímos ainda um conjunto de clientes sem fio que estão configurados para conectar à rede sem fio formada pelos roteadores *Ethanol*.

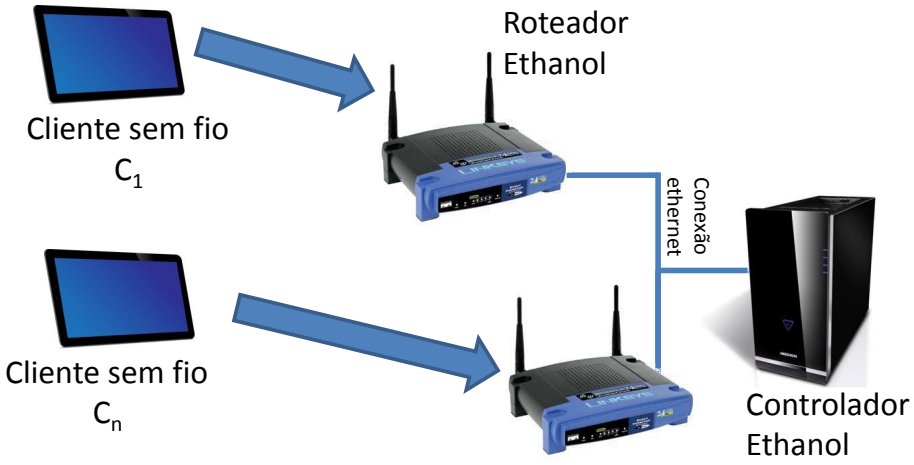


Figura 5.10: Esquema de conexão para experimento de controle de associação

Cada cliente é colocado ao alcance dos dois pontos de acesso sem fio. Ao longo do experimento, os clientes são conectados manualmente ao ESS, selecionando somente o nome da rede na lista fornecida pelo seu software de configuração. Após a conexão, algumas estações foram manualmente desconectadas para que tentassem realizar nova conexão à rede.

5.3.2 Resultados

A figura 5.11 mostra a distribuição das conexões das estações sem fio entre os dois pontos de acesso utilizados no experimento. Cada uma das linhas traçadas no gráfico representa o número de conexões ativas em um dos pontos de acesso controlados. O eixo das abscissas representa uma conexão de um ponto de acesso ao controlador solicitando uma autorização de conexão.

A primeira etapa do experimento, representada pelo intervalo $0 \leq X \leq 13$, consiste na ativação das estações sem fio. Neste período, somente um dos roteadores *Ethanol* está funcionando (AP_2), representado pela linha tracejada em azul. Vemos neste período um comportamos similar ao processo de associação do protocolo IEEE 802.11. Cada solicitação é respondida com uma autorização, representado pelo crescimento em degraus neste intervalo. Isto ocorre porque há somente um ponto de acesso, portanto não há necessidade de balanceamento de carga.

Entre os valores da sequência de conexões ao controlador indicado no eixo X de $13 \leq X \leq 15$, o segundo roteador *Ethanol* é ativado (AP_1), representado pela linha cheia em vermelho. No período de $15 \leq X \leq 20$, forçamos manualmente a desconexão de um conjunto de estações sem fio, que estavam conectadas ao primeiro ponto de acesso. Estas desconexões podem ser percebidas pelo sentido decrescente do gráfico do ponto de acesso AP_2.

No intervalo de $21 \leq X \leq 27$, as estações desconectadas são manualmente reconectadas. O controlador autoriza a conexão no ponto de acesso AP_1 que possui menos clientes conectados. Até que a quantidade de estações esteja novamente igual nos dois pontos de acesso. No intervalo de $28 \leq X \leq 96$, inicialmente desconectamos algumas estações que estavam associadas ao AP_1 e em seguida realizamos nova conexão à rede para observar se o controlador mantém o equilíbrio entre o número de clientes nos dois pontos de acesso.

Desta forma podemos ver no intervalo de $32 \leq X \leq 62$ que diversas solicitações foram feitas ao controlador e que elas foram negadas. Neste período, ambas as linhas não sofrem transição, permanecendo paralelas ao eixo X, indicando, portanto, que não houve alteração nas conexões ativas. Em uma rede IEEE 802.11 a decisão de associação é feita pela estação sem fio. Assim neste período as estações que buscavam conectar no ponto de acesso com maior quantidade de clientes (AP_2) tinham sua solicitação negada

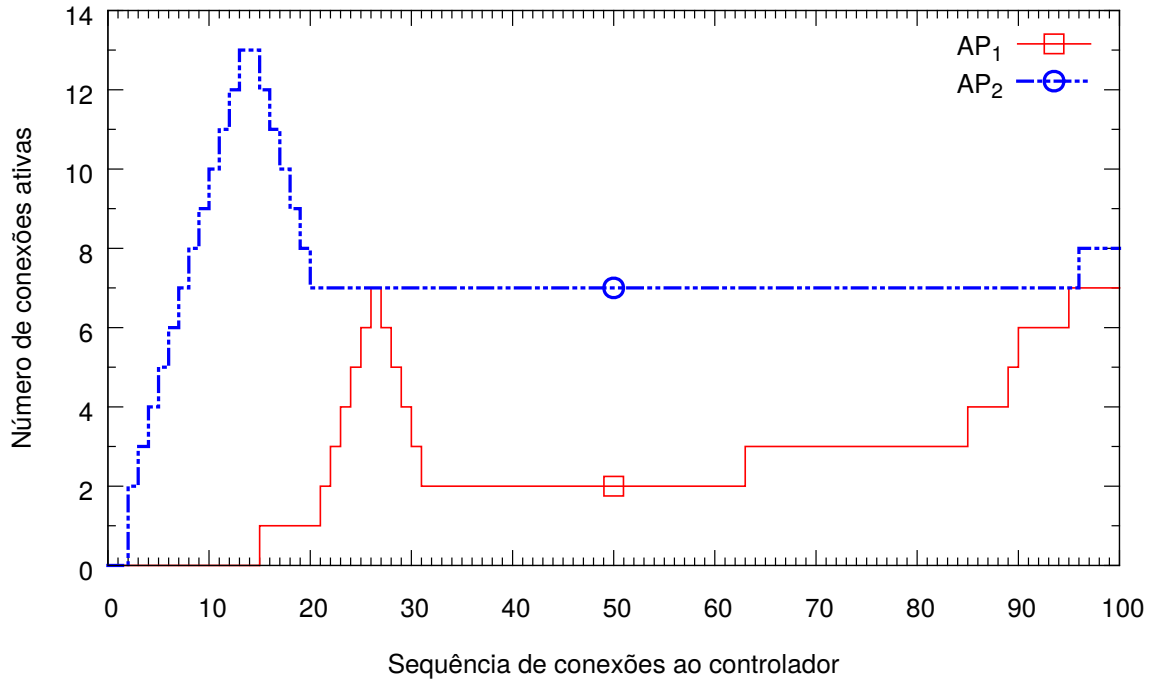


Figura 5.11: Número de estações ativas por ponto de acesso

pelo controlador. Como a quantidade de clientes conectados ao ponto de acesso AP_2 é superior à quantidade de clientes conectados ao outro ponto de acesso, o controlador realiza o balanceamento de carga.

Em $X = 63$, ocorre um pedido para associação ao AP_1. Este pedido é autorizado, pois AP_1 tem o menor número de conexões ativas. As transições verticais, tais como em $X = 63$, indicam que uma estação foi autorizada a conectar ao ponto de acesso representado por aquela linha.

As solicitações que foram realizadas no período de $63 \leq X \leq 96$ para o AP_1 foram autorizadas. O controlador *Ethanol* somente autoriza conexões ao ponto de acesso AP_1 nas próximas tentativas de conexão. Este comportamento está caracterizado pelo crescimento em degraus na figura 5.11 da linha indica a quantidade de clientes em AP_1. Somente quando há novamente o equilíbrio, o controlador permite a conexão no outro ponto de acesso (em $X = 97$).

Medida	**Tempo de execução**	**Intervalo de confiança** $1 - \alpha = 95\%$	
		Mínimo	**Máximo**
Execução local	117,5	106,3	128,7
Execução local com msg p/ controlador	5125,5	3323,2	6927,8

Tabela 5.4: Tempo de processamento de pacotes pelo controlador durante processo de associação (em microssegundos)

A tabela 5.4 mostra o tempo de execução do procedimento de controle do processo de associação do ponto de vista do ponto de acesso. Desta forma os tempos indicados representam o processamento local no ponto de acesso, mais os tempos de envio da mensagem de associação para o controlador, mais o tempo de execução do procedimento *evUserConnecting* no controlador e mais o tempo da mensagem de resposta do controlador para o ponto de acesso. Apresentamos nesta tabela as duas situações: (1) quando o controlador verifica a carga nos pontos de acesso para autorizar as associações (última linha da tabela), e (2) quando o processo de associação é executado somente pelo ponto de acesso, indicado na tabela como **Execução local**.

O procedimento no ponto de acesso, sem alterações, indicado por **Execução local**, consiste somente em atualizar algumas estruturas de dados em memória. Portanto seu tempo de processamento é bastante reduzido. Como podemos ver na tabela 5.4, este tempo médio é da ordem de $0,12\ ms$. Fizemos uma alteração no procedimento do ponto de acesso para que ele enviasse uma solicitação de autorização para o controlador. Esta alteração acrescenta um atraso no processamento. Vemos na tabela 5.4 que existe um acréscimo da ordem de 43 vezes no tempo de execução quando comparado com o procedimento com o controle de associação desabilitado. É importante notar que este acréscimo é da ordem de 5 milissegundos.

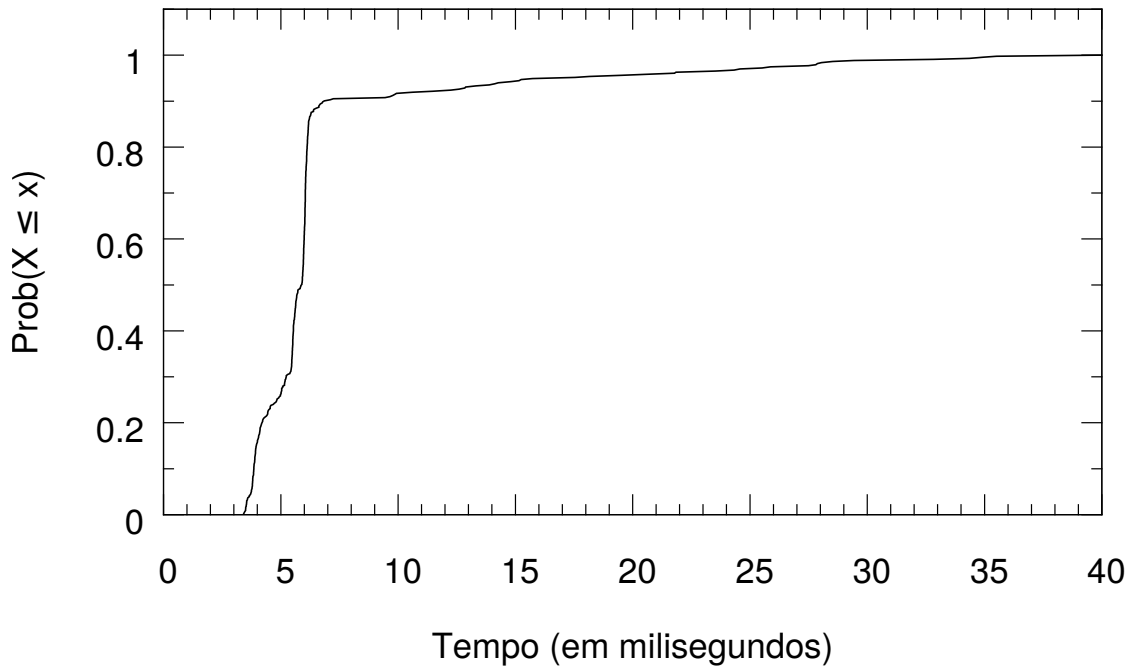


Figura 5.12: Distribuição acumulativa de frequências para os tempos de processamento

Podemos ver na figura 5.12 um gráfico com a distribuição cumulativa do RTT, representado pelo tempo de envio da mensagem pelo ponto de acesso ao controlador, o tempo de processamento no controlador e o tempo da mensagem de resposta. O valor da mediana desta distribuição é de $5,9\ ms$. Desta forma, vemos que o acréscimo identificado decorre principalmente do RTT da comunicação entre o ponto de acesso e o controlador.

5.4 Identificação de interface sem fio falha

Em uma rede sem fio, falhas podem ocorrer em função de problemas de software ou hardware. Estas falhas provocam baixo desempenho na rede, alto tempo de resposta e, até mesmo, desconexão parcial da rede. Um dos tipos de falha que pode ocorrer em um sistema sem fio consiste em problemas de transmissão, sem que o dispositivo (roteador) tenha parado de responder, como por exemplo, pode haver um mal contato na antena.

Com o controlador *Ethanol* podemos identificar falhas na rede sem fio. O controlador pode realizar análises baseadas em dados históricos ou instantâneos, bem como pode obter informações sobre o funcionamento de um ponto de acesso mediante medições realizadas por estações ou por outros pontos de acesso. Em uma rede IEEE

802.11, onde o protocolo IEEE 802.11k está implementado, é possível que uma estação ou ponto de acesso A solicite a outra estação ou ponto de acesso B a lista dos pontos de acesso que B pode receber em um ou mais canais determinados, mediante utilização de varreduras ativas ou passivas, obtendo o relatório denominado "*Beacon report*" (IEEE802.11k [2008]). O controlador *Ethanol* pode regularmente fazer com que seus roteadores solicitem às estações conectadas que enviem esta informação.

Caso o protocolo IEEE 802.11k não esteja implementando, mesmo assim, o controlador pode programar seus pontos de acesso para que cada um faça uma varredura periódica na sua vizinhança. Por meio destas varreduras é possível identificar se uma interface de rede de um ponto de acesso AP_d, ao alcance do AP_x, parou de funcionar. Basta que AP_x não mais detecte as transmissões de *beacon* do AP_d. Assim detectamos uma falha na interface sem fio de AP_d, mesmo que sua interface Ethernet ainda esteja funcionando e que o processo de *hostapd* esteja rodando no ponto de acesso.

O controlador pode determinar quais pontos de acesso não estão funcionando pela comparação da informação fornecida por seus pontos de acesso com a informação referente a quais pontos de acesso estão conectados ao controlador *Ethanol* via OpenFlow. O algoritmo para isto é apresentado no quadro **Algoritmo 4**.

Algoritmo 4 Identificação de falha na interface sem fio de pontos de acesso

1: **function** evBeaconFrameReceived(AP, DetectedAP) ▷
2: recebe os eventos de *beacon* dos APs, informando o AP detectou
3: $t \leftarrow currentTime()$
4: $inRange \leftarrow inRange \bigcup DetectedAP_t$ ▷ insere na lista com um horário de registro
5: $inRange \leftarrow inRange \bigcup AP_t$
6: **end function**
7: **function** detectFaultyAps( ) ▷
8: executado a intervalos regulares (VERIFY_TIME)
9: $inRange \leftarrow \{x_t \mid x_t \in inRange \ \& \ t + EXPIRATION\_TIME \leq currentTime()\}$ ▷ Remove da lista inRange os elementos que já expiraram
10: $N \leftarrow AccessPoint.getVAPs() - isolatedAPs$ ▷ N corresponde a todos os APs detectáveis
11: $N' \leftarrow \{inRange \cap AccessPoint.getVAPs()\} - isolatedAPs$ ▷ APs detectados
12: $faulty \leftarrow N - N'$ ▷ diferença corresponde aos APs não detectados, mas que deveriam ser
13: **return** $faulty$
14: **end function**

O controlador pode receber uma lista com os pontos de acesso ao alcance de um roteador *Ethanol* fazendo uso do evento gerado quando um ponto de acesso recebe uma mensagem de gerenciamento $evMgmtFrameReceived()$. O controlador pode registrar

junto ao ponto de acesso para receber informações toda vez que o ponto de acesso recebe um *beacon*. Ao receber uma mensagem de *beacon*, o roteador *Ethanol* envia ao controlador as informações sobre o ponto de acesso detectado em sua vizinhança.

O procedimento *evBeaconFrameReceived* apresentado no **Algoritmo 4** realiza o tratamento deste evento para mensagem de *beacon*. Ele recebe como parâmetros dois valores: o endereço de hardware do roteador *Ethanol* que realizou a detecção, que denominamos AP, e o endereço de hardware do ponto de acesso detectado ($DetectedAP$). Estes valores são armazenados na lista *inRange*. Cada valor ao ser inserido na lista *inRange* recebe um marcação relativa ao horário que foi detectado. Esta marcação é utilizada para controlar o tempo que este valor permanecerá na lista. Este tempo é controlado por um parâmetro de configuração do controlador denominado $EXPIRATION\_TIME$. O conjunto *inRange* representa todos os pontos de acesso detectados na vizinhança da rede sem fio controlada.

O procedimento *detectFaultyAps* é executado a cada $VERIFY\_TIME$ segundos, sendo chamado por uma interrupção de relógio. Na linha 9 são removidos do conjunto *inRange* todos os pontos de acesso que foram inseridos a mais que $EXPIRATION\_TIME$ segundos.

Além das estruturas de dados já descritas, o controlador precisa de uma lista adicional. O conjunto *isolatedAPs* contém uma lista estática programada pelo administrador de rede contendo todos os pontos de acesso da sua rede sem fio que não estão ao alcance de outros pontos de acesso desta rede. Esta lista é necessária pois em uma implementação real podem haver pontos de acesso instalados que não estejam ao alcance de nenhum outro. Um ponto de acesso que não é detectado por outros ainda pode ser verificado periodicamente. Basta que verifiquemos se sua interface de rede sem fio detecta algum sinal de outros pontos de acesso não pertencentes à rede *Ethanol* ou de estações sem fio. Este procedimento é aplicável a todos os roteadores controlados. Esta verificação é feita na linha 5 do **Algoritmo 4**.

5.4.1 Configuração do experimento

Para testar o funcionamento deste algoritmo montamos um experimento com 3 roteadores *Ethanol*. Estes roteadores estão conectados ao controlador via rede Ethernet, como mostrado na figura 5.13. Os roteadores são ligados e transmitem regularmente informações de escaneamento de *beacons* para o controlador. Durante os experimentos suas antenas são desconectadas ou sua interface de rede sem fio é desligada.

Este experimento foi executado com: (1) o procedimento de processamento de *beacon* no ponto de acesso sem alteração e (2) o procedimento com as alterações neces-

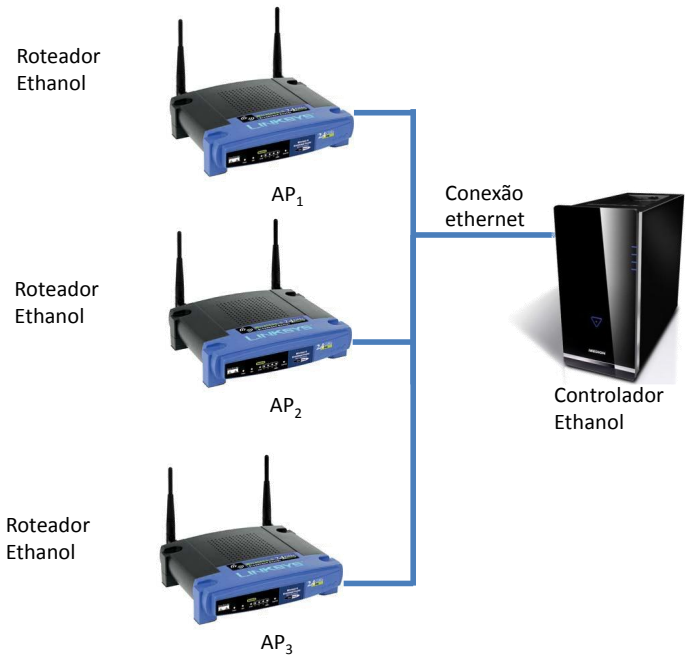


Figura 5.13: Esquema de conexão para experimento de verificação de interfaces sem fio

sárias para o envio das informações do *beacon* para o controlador. Realizamos ainda a medição na condição (2) em duas versões: (1) "sem *thread*", para uma versão do procedimento onde as mensagens para o controlador eram enviadas pelo procedimento de tratamento do *beacon* seguindo o fluxo normal do programa *hostapd*; e (2) "com *thread*", onde as medições são realizadas com o algoritmo modificado para que a mensagem fosse enviada para o controlador por uma *thread*. A comunicação usando XML-RPC é síncrona, desta forma a utilização desta segunda versão permite ganhos de desempenho no ponto de acesso. Isto ocorre porque o procedimento no AP não tem que esperar que o processo de comunicação com o controlador termine.

5.4.2 Resultados

Na figura 5.14 mostramos, ao longo do tempo de funcionamento do experimento, a quantidade de pontos de acesso conectados ao controlador *Ethanol*, a quantidade de pontos de acesso detectados no ambiente e a quantidade de pontos de acesso *Ethanol* que não foram detectados em função de falhas. A figura 5.15 mostra o mesmo experimento, porém apresentamos somente os dados relativos aos roteadores *Ethanol* a fim de facilitar a visualização do comportamento do processo de detecção.

Vemos, nos primeiros 20 segundos, que os 3 pontos de acesso são ligados um após o outro. Estes pontos de acesso realizam, durante sua inicialização, dois processos:

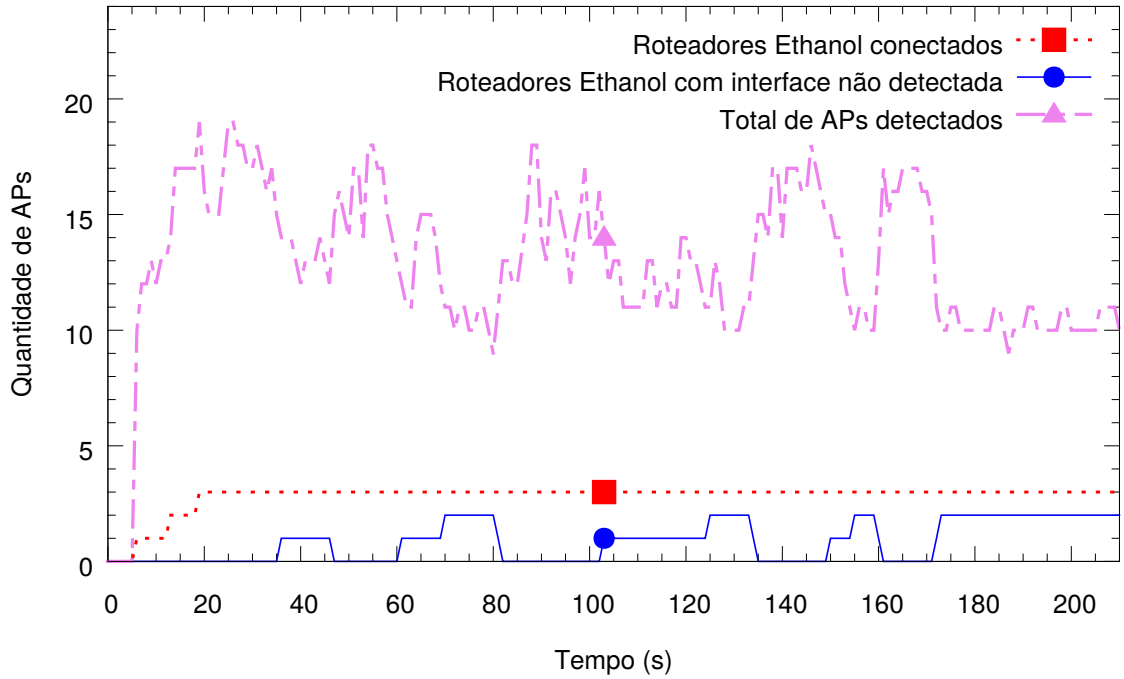


Figura 5.14: Detecção de pontos de acesso sem fio por roteadores *Ethanol*

(1) realizam a conexão utilizando o protocolo OpenFlow ao nosso controlador e (2) informam ao mesmo controlador, via uma mensagem que denominamos "*hello*", via protocolo *Ethanol*, que sua interface sem fio está pronta. Percebemos no gráfico que as interfaces sem fio dos pontos de acesso estão funcionando. Chegamos a esta conclusão pois a linha "Roteadores Ethanol conectados" apresenta um valor não nulo e a linha "Roteadores Ethanol com interface não detectada" é igual a zero.

As falhas na figura 5.15 foram criadas pela desconexão das antenas externas dos adaptadores de rede sem fio. Após a desconexão, as antenas foram reconectadas manualmente. Aos 36 segundos, o controlador identifica que um dos pontos de acesso não pode mais ser detectado. O mesmo ocorre aos 61 segundos. Aos 70 segundos, um segundo ponto de acesso deixa de ser detectado. Após reconectamos as antenas, aos 82 segundos, todos os três pontos de acessos são identificados pelo controlador como funcionando novamente. Repetimos este procedimento mais 3 vezes. Desta forma, observamos que o controlador é capaz de detectar problemas nas interfaces sem fio dos pontos de acesso que gerencia.

Medimos os tempos de execução do procedimento de processamento de *beacon* no ponto de acesso. Para o algoritmo sem *thread*, o tempo de execução médio passou de 12 μs para $3,3$ ms. Neste experimento, podemos reduzir o tempo gasto no processamento do *beacon*, como mostrado na figura, gerando uma *thread* para envio dos dados

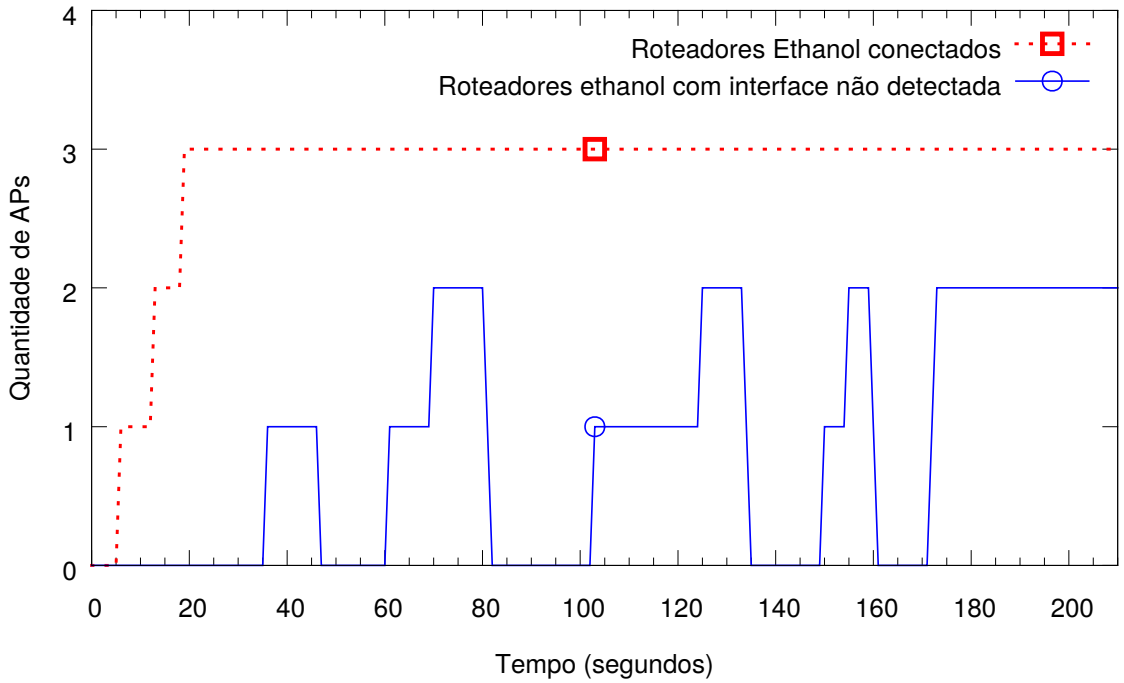
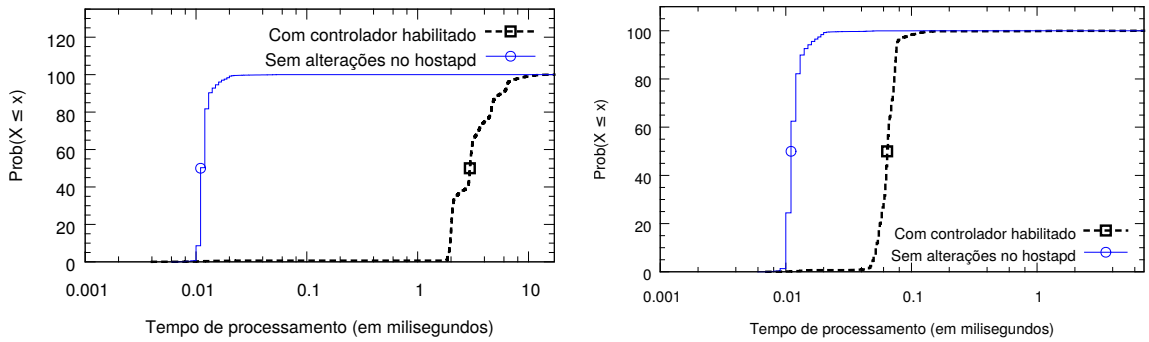


Figura 5.15: Detecção de falhas em interface sem fio em roteadores *Ethanol*



(a) Sem *thread* (b) Com *thread*

Figura 5.16: Distribuição cumulativa dos tempo de processamento de *beacons*

para o controlador, uma vez que não é necessário esperar uma resposta deste. Esta alteração reduz o tempo de execução para $68,5\ \mu s$. Isto representa um enorme ganho se comparado com a mesma condição sem *thread*.

O tempo de execução apresenta um acréscimo compatível com o envio da mensagem mais o processamento no controlador, como apresentado na figura 5.16a. O valor para o tempo que representa 50% das medidas é de $2,96\ ms$ e para 95% é de aproximadamente $6,46\ ms$.

A alteração do procedimento para o envio da mensagem via *thread* melhorou o desempenho. Na figura 5.16b, vemos que 95% do valores são iguais ou inferiores a

77 μs. O algoritmo sem alterações, isto é, sem envio de *beacons* para o controlador é de 16 μs. Esta diferença é explicada pela necessidade de preparar os dados para envio e para a criação da *thread*. Contudo o valor do tempo de processamento com *thread* é 5,7 vezes superior ao tempo de funcionamento normal do procedimento de tratamento de *beacon* sem alteração, para o valor médio, enquanto é cerca de 49 vezes menor que o valor do tempo medido para o algoritmo sem a *thread*.

A fim de determinar o tempo de detecção de falha pelo controlador, montamos o seguinte experimento. Criamos um programa que desliga a intervalos regulares a interface de rede sem fio do roteador. Este programa registra o horário, em microsegundos, que foi solicitado o desligamento. Ele aguarda um período suficiente para a detecção pelo controlador, definido manualmente como 2 segundos, e logo depois faz o religamento da interface do roteador.

Uma entrada na estrutura de dados *inRange* do **Algoritmo 4** é mantida por um intervalo de $EXPIRATION\_TIME$ segundos. O algoritmo *detectFaultyAps* é executado a cada $VERIFY\_TIME$ segundos. Coletamos os tempos usando quatro combinações dos valores de configuração, como mostrado na figura 5.5. A primeira combinação foi utilizada nos experimentos que realizamos neste estudo de caso. Abaixo de $VERIFY\_TIME = 50\ ms$, começamos a ter problema de sincronização de processos, pois a atualização da estrutura *inRange* é realizada em dois procedimentos diferentes - um que responde ao evento de *beacon* recebido pelo roteador e outro que remove os valores de *inRange* que já expiraram.

#	**Parâmetros de configuração do algoritmo**		**Valor médio observado** τ_f	**Desvio Padrão**	**Intervalo de confiança** $1-\alpha = 95\%$	
	Tempo de verificação (segundos) $VERIFY\_TIME$	**Tempo de expiração (segundos)** $EXPIRATION\_TIME$			**Valor mínimo**	**Valor máximo**
1	0.5	1	1,206	0,178	1,185	1,228
2	0,1	0,5	0,498	0,070	0,489	0,507
3	0,05	0,2	0,176	0,050	0,169	0,182
4	0,05	0,05	0,051	0,026	0,047	0,055

Tabela 5.5: Verificação dos efeitos dos parâmetros de configuração sobre o tempo de detecção de falhas pelo controlador

Na figura 5.17 vemos uma linha de tempo exemplificando o processo de detecção. Para efeito de simplificação, consideramos que $EXPIRATION\_TIME$ é múltiplo de $VERIFY\_TIME$. No exemplo, consideramos $EXPIRATION\_TIME = 3 \times VERIFY\_TIME$. Desta forma, vamos considerar que o *beacon* do AP em questão foi detectado em $t = 0$, assim este valor será retirado da lista *inRange* em $t > 3$, ou seja, em $t = 4$. Se este AP parar de transmitir o *beacon* no intervalo de $0 < t < 4$,

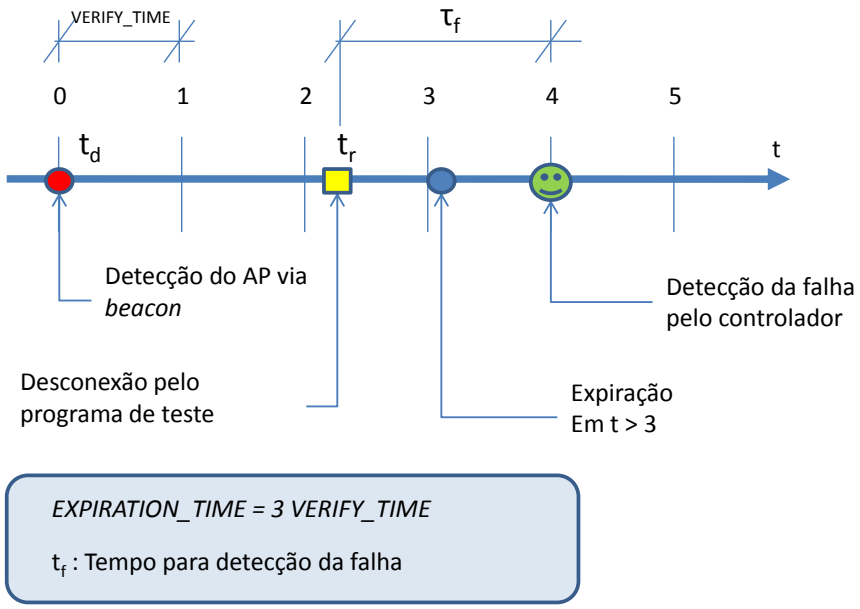


Figura 5.17: Linha de tempo para detecção de falhas

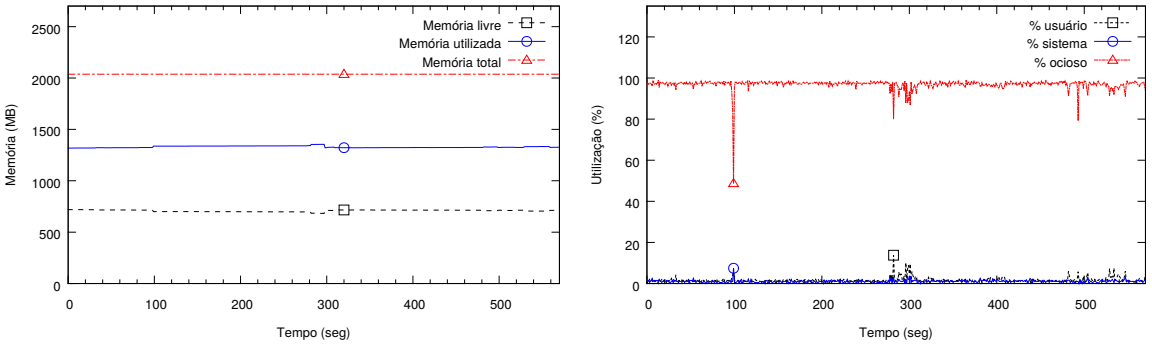
a marcação de horário deste item na lista *inRange* não será atualizada[3]. Portanto, em $t = 4$, o AP será retirado da lista e o controlador irá detectar que ele falhou em um tempo τ_f.

Desta forma, o menor tempo de detecção ocorre quando o ponto de acesso é desconectado um pouco antes do horário de expiração do último *beacon* que atualizou *inRange*, ficando o atraso da ordem de $VERIFY\_TIME$. Enquanto o maior tempo de detecção ocorre quando o ponto de acesso falha logo depois da sua entrada em *inRange* ter sido atualizada. Neste caso o atraso é da ordem de $EXPIRATION\_TIME + VERIFY\_TIME$.

5.5 Uso de memória e CPU nos estudos de caso

Nesta seção avaliamos o uso de memória e CPU no hardware utilizado para implementar o controlador *Ethanol* e o roteador. Foram realizadas medições para definição da linha de base do uso de memória e do uso de processador nos computadores utilizados para rodar o controlador e o roteador *Ethanol*, respectivamente. Em seguida foram obtidos os valores para estas medidas em cada um dos quatro casos de uso. As medições foram realizadas utilizando o programa *sar* versão 10.2.0 (`http://sebastien.godard.pagesperso-orange.fr/`) do pacote *sysstat*. Estes resultados são apresentados nas

[3]Se um *beacon* for detectado no intervalo $0 < t < 4$, o tempo de expiração é atualizado e, portanto, o AP não será retirado da lista em $t = 4$.



(a) Uso de memória com controlador desativado

(b) Uso de processador com controlador desativado

Figura 5.18: Uso de recursos no computador que executará o controlador

duas subseções seguintes. Terminamos esta seção apresentando de forma gráfica uma comparação entre os diversos casos.

5.5.1 Uso de memória e processador pelo controlador *Ethanol*

Nesta seção apresentamos o uso de memória e CPU no computador utilizado para implementar o controlador *Ethanol*. É importante destacar que os resultados apresentados mostram que a influência dos experimentos sobre o funcionamento do computador não foi muito significativa. Contudo este hardware é aquele que podemos exercer maior influência. O acréscimo de memória é possível de ser efetuada sem maiores problemas dentro dos limites da placa mãe utilizada. O aumento de capacidade computacional e mesmo da quantidade de memória para valores acima de 4GB podem ser conseguidos pela troca da plataforma de hardware.

5.5.1.1 Linha de base

A figura 5.18 mostra a linha de base para o uso de memória e para uso de processador no computador preparado para rodar o controlador *Ethanol*. Utilizamos estes valores para comparar o efeito do uso do software desenvolvido para os estudos de caso sobre a memória e sobre o processador. Nas avaliações, foram calculados o valor médio e o desvio padrão das grandezas medidas, por exemplo o percentual utilizado da memória RAM, bem como os valores mínimo e máximo observados durante o levantamento. Foi calculado ainda o intervalo de confiança para o valor médio, considerando um coeficiente de confiança $(1 - \alpha) = 95\%$, que determina o quanto estas estimativas são prováveis.

Medida	Valor médio	Intervalo de confiança $1-\alpha = 95\%$		Desvio Padrão	Valor mínimo observado	Valor máximo observado
		Mínimo	Máximo			
Livre(MB)	691,6	690,9	692,2	8,3	666,9	702,8
Usada(MB)	1298,1	1297,4	1298,8	8,3	1286,9	1322,8
% Usado	65,24	65,21	65,28	0,42	64,68	66,48

Tabela 5.6: Uso de memória no computador com controlador desativado

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$		Desvio Padrão	Valor mínimo observado	Valor máximo observado
		Mínimo	Máximo			
% usuário	1,69	1,59	1,79	1,22	0,00	13,78
% sistema	1,07	1,03	1,12	0,57	0,00	7,46
% ocioso	96,91	96,68	97,13	2,72	48,76	99,49

Tabela 5.7: Uso de CPU no computador com controlador desativado

Como podemos ver na tabela 5.6, o computador já está com cerca de 65% de sua memória RAM utilizada, totalizando cerca de 1298 MB da memória RAM utilizada (692 MB de memória livre). Este consumo de memória deve-se à existência de diversos serviços e aplicativos neste servidor, como serviço de DHCP (*Dynamic Host Configuration Protocol*) e o uso de interface gráfica *Gnome Desktop*. Pela tabela 5.7, observamos que o processador do computador não está sendo muito demandado, ficando, em média, ocioso por mais de 96% do tempo. Os processos de usuário representam em média 1,7% do tempo de processador, atingindo o máximo de 13,78%. Vemos ainda que o consumo dos processos de sistema também é baixo, chegando em média a 1,07%.

5.5.1.2 Controle de transmissões ARP

Realizamos durante o experimento a coleta de diversos parâmetros de uso de memória e processador, utilizando o aplicativo *sar*. Podemos observar pela tabela 5.8 que o valor médio de utilização do processador no controlador ficou, em média, na faixa de 6,29% a 6,75%. Se compararmos com nossa linha de base, na tabela 5.7, vemos que o tempo do uso de processador pelos processos de usuário passa de 1,69% (na linha de base) para 6,52% (durante o experimento). Os valores máximo e mínimo observados mostram que o processador do controlador foi pouco demandado durante todo o experimento.

Com relação ao consumo de memória durante o experimento, observamos, na

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$		Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
		Mínimo	Máximo				
% usuário	6,52	6,29	6,75	2,18	0,00	11,50	285,8%
% sistema	2,22	2,13	2,31	0,83	0,00	5,00	107,5%
% ocioso	91,03	90,75	91,30	2,61	84,50	98,50	-6,1%

Tabela 5.8: Uso de CPU no controlador durante o experimento de controle ARP

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$ Mínimo	Máximo	Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
Livre(MB)	146,8	145,9	147,8	9,2	130,9	159,5	-78,8%
Usada(MB)	1843	1842	1843,9	9,2	1830,3	1858,9	42,0%
% Usado	92,62	92,57	92,67	0,46	91,99	93,42	42,0%

Tabela 5.9: Uso de memória no controlador durante o experimento de controle ARP

tabela 5.9, que, em média, o computador ficou com 92,62% de sua memória RAM utilizada. O uso de memória é relativamente constante durante o período do experimento. Os valores mínimo e máximo para o percentual de uso de memória (%Usado) são, respectivamente, 91,99% e 93,42%, menos de 2% de variação total. Se compararmos com a linha de base na tabela 5.6 com o valor médio da tabela 5.9, vemos que o consumo médio de memória RAM passa de 65,2% para 92,62%. Este acréscimo de 42% deve-se ao carregamento de nosso programa, suas estruturas de dados, do POX e suas estruturas e do Python.

5.5.1.3 Implementação de QoS para tráfego Ethernet e sem fio

Durante o experimento detectamos pequenas variações no uso de memória no controlador, sendo que o valor mínimo e máximo para o uso de memória foram de 71,36% e 75,40%, respectivamente. Pela tabela 5.10 observamos que o valor médio de utilização da memória RAM foi da ordem de 73,9%. Desta forma, comparando com a linha de base da tabela 5.6, o uso de memória foi acrescido de menos de 15% com a utilização do controlador durante o experimento de controle de fluxos em relação à linha de base.

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$ Mínimo	Máximo	Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
Livre(MB)	519,6	517,5	521,8	20,6	489,5	569,9	-24,9%
Usada(MB)	1470,2	1468	1472,3	20,6	1419,9	1500,3	13,3%
% Usado	73,89	73,78	73,99	1,03	71,36	75,40	13,3%

Tabela 5.10: Uso de memória no controlador durante experimento de controle de fluxos

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$ Mínimo	Máximo	Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
% usuário	9,47	8,67	10,27	7,59	0,51	26,04	460,4%
% sistema	3,77	3,56	3,97	1,94	1,01	9,60	252,3%
% ocioso	86,34	85,33	87,35	9,59	51,81	97,94	-10,9%

Tabela 5.11: Uso de CPU no controlador durante experimento de controle de fluxos

Observamos na tabela 5.11 que o uso do processador pelos processos de usuário, dentre os quais estão os processos relacionados ao controlador, ficou em um valor médio

de 9,5%, atingindo um pico máximo de quase 26%. Se compararmos com a linha de base, na tabela 5.7, observamos que o funcionamento do controlador acrescenta em média quase 8 pontos percentuais ao consumo de CPU - cerca de 5,6 vezes o valor sem o controlador.

5.5.1.4 Controle do processo de associação de estações sem fio

Pela tabela 5.12 observamos que o valor médio de utilização da memória RAM foi da ordem de 94,6%, desta forma, comparando com a linha de base da tabela 5.6, o uso de memória foi acrescido de 45% com a utilização do controlador para controle de associação. Durante o experimento verificamos que as variações no uso de memória durante a execução do experimento são pequenas, ficando os valores mínimo e máximo em 94,13% e 95,15%, respectivamente.

Uso de CPU	**Valor médio**	**Intervalo de confiança $1-\alpha = 95\%$**		**Desvio Padrão**	**Valor mín. observado**	**Valor máx. observado**	**Dif. relativa à linha de base**
		Mínimo	**Máximo**				
Livre(MB)	107,1	106,6	107,6	6,4	96,5	116,7	-84,5%
Usada(MB)	1882,6	1882,1	1883,1	6,4	1873,0	1893,2	45,0%
% Usado	94,62	94,59	94,64	0,32	94,13	95,15	45,0%

Tabela 5.12: Uso de memória no controlador durante experimento de controle de associação

Uso de CPU	**Valor médio**	**Intervalo de confiança $1-\alpha = 95\%$**		**Desvio Padrão**	**Valor mín. observado**	**Valor máx. observado**	**Dif. relativa à linha de base**
		Mínimo	**Máximo**				
% usuário	3,92	3,84	4,01	1,03	1,53	8,63	132,0%
% sistema	0,93	0,89	0,97	0,48	0,00	2,96	-13,1%
% ocioso	94,72	94,59	94,86	1,65	75,63	97,95	-2,3%

Tabela 5.13: Uso de CPU no controlador durante experimento de controle de associação

O funcionamento do controle de associação gera pouco processamento durante o experimento, ficando o consumo dos processos de usuário em um valor médio de 3,92% como mostramos na tabela 5.13. O pico máximo de utilização foi de 8,3%, enquanto o valor mínimo observado foi de 1,53%. Se compararmos estes valores com a linha de base, na tabela 5.7, observamos que o funcionamento do controlador acrescenta em média cerca de 7 pontos percentuais ao consumo de CPU para o caso máximo e de menos de 2,3% no caso médio.

5.5.1.5 Identificação de interface sem fio falha

O uso de memória no controlador durante a execução do experimento é praticamente constante, comprovado pela pequena variação nos valores mínimos e máximos apresentados na tabela 5.14. Realizamos as medições do consumo de memória com a

mensagem sendo enviada por *thread* e sem o uso de *thread*. O uso de memória médio saiu de 65,24% da linha de base, na tabela 5.6, para 71,1% para o algoritmo sem *thread* e 69,8% para o algoritmo com *thread*.

Medida		**Valor médio**	**Intervalo de confiança** $1-\alpha = 95\%$		**Desvio Padrão**	**Valor mín. observ.**	**Valor máx. observ.**	**Dif. relativa à linha de base**
			Mínimo	**Máximo**				
Sem thread	Livre(MB)	575,0	574,6	575,4	2,9	571,5	583,4	-16,9%
	Usada(MB)	1414,7	1414,3	1415,1	2,9	1406,3	1418,1	9,0%
	% Usado	71,10	71,08	71,12	0,15	70,68	71,28	9,0%
Com thread	Livre(MB)	601,5	600,2	602,8	8,8	595,8	650,6	-13,0%
	Usada(MB)	1388,2	1386,9	1389,5	8,8	1339,1	1393,9	6,9%
	% Usado	69,77	69,70	69,84	0,44	67,30	70,05	6,9%

Tabela 5.14: Uso de memória no controlador durante experimento de identificação de falhas na interface sem fio

Verificamos que há um acréscimo no uso de CPU, quando comparado com a linha de base. Na linha de base, os processos de usuário consumiam em média 1,69% do tempo de processamento, enquanto no experimento este consumo vai para a faixa de 18%, sendo em média 17,9% para o algoritmo sem o uso de *thread* para envio das mensagens de *beacon* ao controlador e 17,41% para o algoritmo enviando as mensagens de *beacon* ao controlador por *thread*.

Medida		**Valor médio**	**Intervalo de confiança** $1-\alpha = 95\%$		**Desvio Padrão**	**Valor mín. observ.**	**Valor máx. observ.**	**Dif. relativa à linha de base**
			Mínimo	**Máximo**				
Sem thread	% usuário	17,90	17,28	18,52	4,24	0,00	26,50	959,2%
	% sistema	5,14	4,97	5,31	1,15	1,00	8,08	380,4%
	% ocioso	76,73	76,02	77,44	4,85	68,50	99,00	-20,8%
Com thread	% usuário	17,41	16,81	18,02	4,08	0,00	24,24	930,2%
	% sistema	5,39	5,20	5,58	1,30	0,00	10,50	403,7%
	% ocioso	76,98	76,28	77,67	4,68	67,84	100,00	-20,6%

Tabela 5.15: Uso de CPU no controlador durante experimento de identificação de falhas na interface sem fio

5.5.2 Uso de memória e processador pelo roteador *Ethanol*

Nesta seção apresentamos os valores obtidos para o uso de memória e para o uso de processador no computador preparado para rodar o roteador *Ethanol*. São mostrados os valores obtidos para a linha de base bem como os valores medidos em cada um dos experimentos.

5.5.2.1 Linha de base

Os valores para a linha de base para o uso de memória e para o uso de processador no computador preparado para rodar o roteador *Ethanol* são mostrados nesta subseção. Estes valores foram utilizados para comparar o efeito do uso do software desenvolvido para os estudos de caso sobre a memória e sobre o processador no roteador. O computador foi analisado com os serviços de *Openvswitch* e *hostapd* desativados, pois estes são os softwares necessários para execução do *Ethanol*.

Na tabela 5.16 podemos ver que o computador utiliza cerca de 25% de sua memória RAM, totalizando cerca de 1489 MB de memória RAM livre. Já pela tabela 5.17, observamos que o processador do computador não está sendo muito demandado, ficando ocioso por mais de 99,5% do tempo. Os processos de usuário representam em média 0,13% do tempo de processador, atingindo o máximo de 0,5% durante o levantamento. Vemos também que o consumo dos processos de sistema é baixo, chegando em média a 0,26%.

Medida	**Valor médio**	**Intervalo de confiança** $1-\alpha = 95\%$		**Desvio Padrão**	**Valor mínimo observado**	**Valor máximo observado**
		Mínimo	**Máximo**			
Livre(MB)	1489,0	1488,7	1489,2	2,5	1484,6	1493,2
Usada(MB)	504,2	504,0	504,5	2,5	500,0	508,5
% Usado	25,30	25,28	25,31	0,12	25,08	25,51

Tabela 5.16: Uso de memória no computador com roteador desativado

Uso de CPU	**Valor médio**	**Intervalo de confiança** $1-\alpha = 95\%$		**Desvio Padrão**	**Valor mínimo observado**	**Valor máximo observado**
		Mínimo	**Máximo**			
% usuário	0,13	0,11	0,15	0,22	0,00	0,50
% sistema	0,26	0,24	0,29	0,27	0,00	1,01
% ocioso	99,51	99,47	99,55	0,39	97,99	100,00

Tabela 5.17: Uso de CPU no computador com roteador desativado

5.5.2.2 Controle de transmissões ARP

Utilizando o aplicativo *sar*, realizamos a coleta de diversos parâmetros de uso de memória e processador no roteador *Ethanol*, durante o experimento de controle de fluxo ARP. Podemos observar pela tabela 5.18 que o valor médio de utilização do processador no roteador ficou, em média, na faixa de 0,75% a 0,92%. Se compararmos com a linha de base do roteador, na tabela 5.17, vemos que o tempo do uso de processador pelos processos de usuário passa de 0,13% para 0,84% durante o experimento.

O consumo de memória durante o experimento no roteador é mostrado na tabela 5.19. Em média, o consumo de memória RAM durante o experimento no roteador ficou

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$		Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
		Mínimo	Máximo				
% usuário	0,84	0,75	0,92	1,12	0,00	20,60	546,2%
% sistema	1,49	1,45	1,54	0,63	0,00	10,15	473,1%
% ocioso	97,53	97,40	97,66	1,68	69,85	99,50	-2,0%

Tabela 5.18: Uso de processador no roteador durante o experimento de controle ARP

em média igual a 32,68% de sua memória RAM total. O uso de memória é relativamente constante durante o período do experimento. A diferença entre os valores mínimo e máximo observados para o percentual de uso de memória (%Usado) é de 1,03%. Se compararmos com o valor de 25,3% da linha de base na tabela 5.6, vemos que o valor médio durante o experimento sofre um acréscimo de menos de 30%. Este aumento deve-se ao carregamento do *Openvswitch* e *hostapd*.

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$		Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
		Mínimo	Máximo				
Livre(MB)	1341,8	1341,5	1342,2	4,6	1330,8	1351,4	-9,9%
Usada(MB)	651,4	651	651,7	4,6	641,8	662,4	29,2%
% Usado	32,68	32,66	32,70	0,23	32,20	33,23	29,2%

Tabela 5.19: Uso de memória no roteador durante o experimento de controle ARP

5.5.2.3 Implementação de QoS para tráfego Ethernet e sem fio

Ao longo do experimento foram detectadas pequenas variações no uso de memória no roteador. Os valores mínimo e máximo para o uso de memória foram, respectivamente, 702,2 MB e 737,9 MB, representando uma variação total de menos de 1,8%, como podemos ver na tabela 5.20. O valor médio de utilização da memória RAM foi de 36,16%. Assim, comparando com a linha de base da tabela 5.16, o uso de memória foi cerca 43% superior à linha de base, contudo isto representa somente um acréscimo de 10,86%, em média, do uso total de memória disponível no roteador durante o controle de fluxos.

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$		Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
		Mínimo	Máximo				
Livre(MB)	1272,4	1271,7	1273,1	8,9	1259,9	1291	-14,5%
Usada(MB)	720,8	720	721,5	8,9	702,2	733,3	43,0%
% Usado	36,16	36,12	36,20	0,44	35,23	36,79	42,9%

Tabela 5.20: Uso de memória no roteador durante experimento de controle de fluxos

O valor médio para o consumo de CPU no roteador durante o experimento foi de 0,59%, como podemos observar na tabela 5.21. Ao compararmos com o valor médio

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$		Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
		Mínimo	Máximo				
% usuário	0,59	0,55	0,63	0,48	0,00	3,52	353,8%
% sistema	0,53	0,50	0,56	0,38	0,00	2,49	103,8%
% ocioso	98,65	98,58	98,73	0,87	91,46	100,00	-0,9%

Tabela 5.21: Uso de CPU no roteador durante experimento de controle de fluxos

da linha de base, na tabela 5.17. Este acréscimo médio é de cerca de 4,5 vezes. O consumo de processador pelos processos de usuário durante o experimento chegou ao limite 3,52% no roteador.

Desta forma notamos não haver uma sobrecarga muito significativa no uso de memória e processador em função da utilização do software do *Ethanol* no roteador, durante os experimentos. Não realizamos contudo um teste de sobrecarga no roteador para verificar qual é o volume de tráfego suportado pelo roteador em função destes dois parâmetros medidos.

5.5.2.4 Controle do processo de associação de estações sem fio

O valor médio de utilização da memória RAM foi da ordem de 26,4%, como podemos observar na tabela 5.22. Comparando com a linha de base da tabela 5.16, vemos que o uso de memória foi pouco modificado. Na linha de base, o uso médio de memória é de 25,3%, desta forma observamos que há um acréscimo de 4,4% com a utilização do roteador para controle de associação. Isto representa um acréscimo de 1,1% no consumo da memória total no roteador. Durante o experimento verificamos que as variações no uso de memória são menores que 1%, desta forma o valor característico do uso de memória é praticamente constante.

O consumo de CPU dos processos de usuário fica em um valor médio de 5,46% como mostramos na tabela 5.23. A amplitude do intervalo de confiança para $1 - \alpha = 95\%$ é somente de 1%. Desta forma o controle de associação gera um acréscimo de processamento durante o experimento que é quase constante, contudo observamos um pico de utilização foi de 22% (máximo). Se compararmos estes valores com a linha de base, na tabela 5.17, observamos que o consumo de CPU, no caso

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha = 95\%$		Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
		Mínimo	Máximo				
Livre(MB)	1467,0	1466,5	1467,5	4,4	1460,9	1476,3	-1,5%
Usada(MB)	526,2	525,7	526,7	4,4	516,9	532,3	4,4%
% Usado	26,4	26,37	26,42	0,22	25,93	26,7	4,4%

Tabela 5.22: Uso de memória no roteador durante experimento de controle de associação

Uso de CPU	Valor médio	Intervalo de confiança $1-\alpha=95\%$		Desvio Padrão	Valor mín. observado	Valor máx. observado	Dif. relativa à linha de base
		Mínimo	Máximo				
% usuário	5,46	5,26	5,67	1,73	0,00	22,11	4100,0%
% sistema	2,91	2,80	3,02	0,95	0,50	7,04	1019,2%
% ocioso	91,49	91,22	91,75	2,25	70,85	98,50	-8,1%

Tabela 5.23: Uso de CPU no roteador durante experimento de controle de associação

médio, sai de 0,13% para 5,46%. Isto representa uma variação considerável (cerca de 42 vezes maior). Contudo ainda representa um consumo total de CPU do roteador relativamente baixo.

5.5.2.5 Identificação de interface sem fio falha

O uso de memória no roteador varia menos de 1% durante a execução do experimento em relação aos valores mínimos e máximos. Vemos na tabela 5.24 que os valores mínimos e máximos variam pouco. A amplitude dos valores medidos, para o funcionamento com o envio de mensagens de *beacon* para o controlador utilizando ou não de *thread*, é menor que 1%.

Na linha de base, o uso de memória médio é de 25,3% (tabela 5.16). Quando o módulo *hostapd* funciona sem o uso de *thread* para envio de mensagens de *beacon* ao controlador, o consumo de memória no roteador fica em média em 28,74%, ficando em 26,64% quando as *threads* são utilizadas.

Medida		Valor médio	Intervalo de confiança $1-\alpha=95\%$		Desvio Padrão	Valor mín. observ.	Valor máx. observ.	Dif. relativa à linha de base
			Mínimo	Máximo				
Sem thread	Livre(MB)	1420,4	1420	1420,8	4,9	1412,2	1430,9	-4,6%
	Usada(MB)	572,8	572,4	573,2	4,9	562,3	580,9	13,6%
	% Usado	28,74	28,72	28,76	0,24	28,21	29,15	13,6%
Com thread	Livre(MB)	1462,1	1461,7	1462,5	4,9	1453,5	1471,1	-1,8%
	Usada(MB)	531,1	530,7	531,5	4,9	522,1	539,7	5,3%
	% Usado	26,64	26,62	26,66	0,24	26,19	27,07	5,3%

Tabela 5.24: Uso de memória no roteador durante experimento de identificação de falhas na interface sem fio

Verificamos que há um acréscimo no uso de CPU, quando comparado com a linha de base mostrada na tabela 5.17, de mais de 1500%. Na linha de base, o consumo de CPU pelos processos de usuário foi de 0,13% em média. Este valor chega a 2,52% para a execução do módulo *hostapd* funcionando com o envio de mensagens de *beacon* por *thread* para o controlador e a 2,12% para o módulo sem *thread*. Vemos que apesar desta grande variação, não existe uma grande sobrecarga, em valores absolutos, no computador que funciona como roteador *Ethanol* em quaisquer dos casos.

Medida		Valor médio	Intervalo de confiança $1-\alpha = 95\%$		Desvio Padrão	Valor mín. observ.	Valor máx. observ.	Dif. relativa à linha de base
			Mínimo	Máximo				
Sem thread	% usuário	2,12	2,07	2,17	0,82	0,00	4,48	1530,8%
	% sistema	0,91	0,88	0,93	0,48	0,00	2,94	250,0%
	% ocioso	96,79	96,73	96,86	1,08	91,46	99,49	-2,7%
Com thread	% usuário	2,52	2,46	2,57	0,92	0,0	6,28	1838,5%
	% sistema	1,06	1,03	1,10	0,56	0,0	4,41	307,7%
	% ocioso	96,21	96,13	96,28	1,25	88,38	100,00	-3,3%

Tabela 5.25: Uso de CPU no roteador durante experimento de identificação de falhas na interface sem fio

5.5.3 Comparação

Nesta seção apresentamos uma comparação do consumo (em percentual) de memória e processador no controlador e no comutador, mostrando os valores relativos à linha de base para os quatro estudos de caso. Os gráficos mostram no eixo das abscissas o percentual de consumo relativo de processador pelos processos de usuário e no eixo das ordenadas o percentual de consumo relativo de memória. São apresentados ainda os intervalos de confiança para os valores nos eixos X e Y. Contudo os intervalos de confiança, principalmente no eixo Y, são muito pequenos. Desta forma o símbolo que identifica o ponto sobrepõe-se ao intervalo. Somente em alguns casos é possível identificar o intervalo de confiança para o consumo percentual relativo de CPU (eixo X).

Na figura 5.19 são apresentados os valores para o controlador. Os valores formam uma aglomeração próxima ao valor da linha de base, apresentando uma dispersão máxima de cerca de 30 pontos percentuais em torno o valor da linha de base no eixo Y e 18 pontos percentuais em torno o valor da linha de base no eixo X. Observamos que o experimento de detecção de falhas apresentou o maior consumo de processador entre os estudos de caso. Este comportamento ocorre pois o programa no controlador é chamado com maior frequência, no caso uma vez a cada *beacon* detectado pelos pontos de acesso. Os experimentos de controle de associação e de fluxos ARP são os que apresentam maior consumo de memória. Este comportamento é esperado para o controle de fluxo ARP, uma vez que neste experimento armazenamos o mapeamento dos diversos fluxos, portos e endereços. No controle de associação, a estrutura de dados gerada pela mensagens contém diversas informações adicionais sobre os pontos de acesso detectados, desnecessárias para o experimento, porém que estão armazenadas em memória. Em função disto, o consumo de memória foi mais significativo.

Na figura 5.20 apresentamos o consumo relativo de memória e CPU para o roteador. Os valores apresentados são portanto o incremento em relação à linha de base,

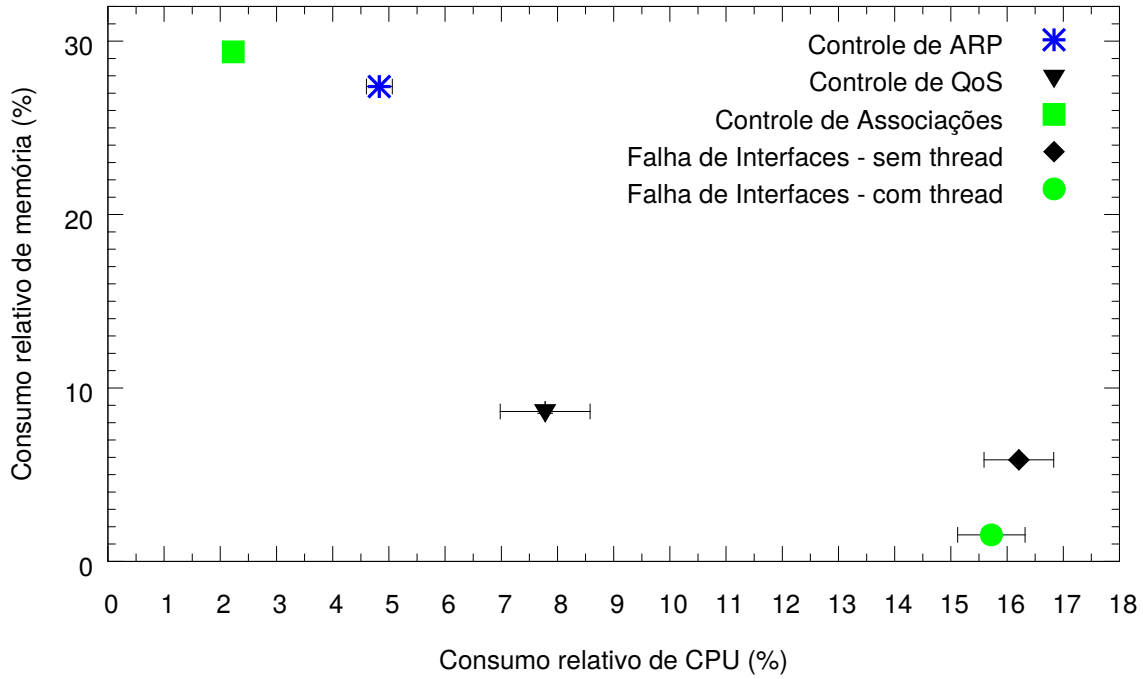


Figura 5.19: Comparação de uso relativo de memória e processador nos estudos de casos do controlador

mostrada na seção 5.5.2.1. Vemos que no roteador os valores formam uma aglomeração próxima ao valor da linha de base, não gerando uma sobrecarga significativa. A aglomeração dos valores apresenta menor dispersão do que no controlador. Observamos ainda que os experimentos de detecção de falhas e controle de associações apresentam uma tendência para maior uso de processamento, enquanto os experimentos com controles de QoS e controle de fluxo ARP geram um consumo maior de memória.

5.6 Conclusão

Vimos pelos resultados apresentados para os quatro casos que: (a) o *Ethanol* permite o controle do fluxo de transmissões de quadros ARP entre as redes Ethernet e rede local sem fio; (b) a arquitetura permite realizar o controle de fluxo de pacotes de estações da rede Ethernet e da rede sem fio mediante utilização de filas de prioridade; (c) é possível realizar um balanceamento de carga nos pontos de acesso mediante controle do processo de associação das estações aos pontos de acesso sem fio; e (d) podemos detectar falhas na interface de rede sem fio, utilizando os recursos fornecidos pelo *Ethanol*. Os casos apresentados neste capítulo são somente alguns exemplos de aplicações da arquitetura proposta. Outros podem ser feitos de acordo com a necessidade de cada rede.

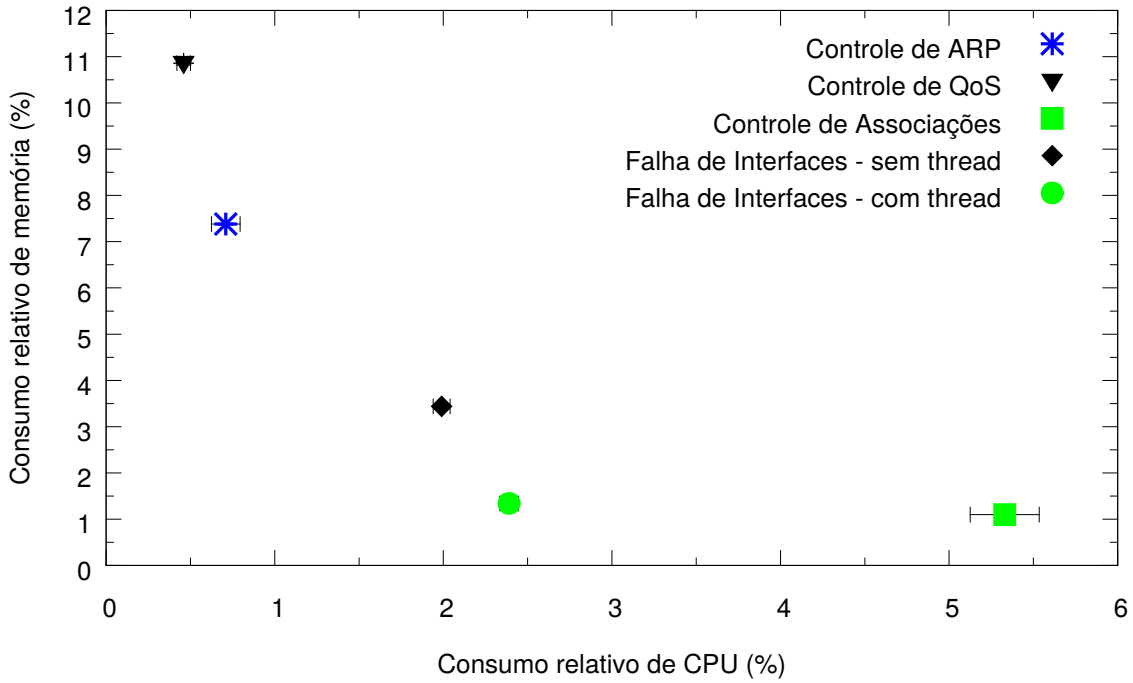


Figura 5.20: Comparação de uso relativo de memória e processador nos estudos de casos do roteador

Os casos apresentados poderiam ser executados em uma plataforma de hardware de baixo custo. Os resultados de desempenho mostrados na seção 5.5.2 corroboram esta afirmação, contudo não possuimos parâmetros para realizar uma transposição dos resultados medido nos nossos experimentos para a realidade de um ponto de acesso de baixo custo. Esta avaliação fica como sugestão para trabalhos futuros.

A utilização de uma abordagem SDWN como no caso do *Ethanol*, e tipicamente feita com controladoras comerciais e pontos de acesso especializados, permite a redução do *CAPEX*, isto é, do investimento. Ao mesmo tempo proporciona benefícios superiores, ao permitir que o administrador de rede possa criar soluções sob medida para sua necessidade.

Capítulo 6

Conclusões e Trabalhos Futuros

Nesta dissertação exploramos uma nova arquitetura para gerenciamento de redes locais sem fio definidas pelo padrão IEEE 802.11 utilizando os conceitos de redes definidas por software, denominada *Ethanol*. Com o *Ethanol* pretendemos atender a muitos dos problemas existentes no gerenciamento de redes sem fio e também em sua configuração. Esta é uma arquitetura aberta que permite ao administrador de redes implementar funções sob medida para suas necessidades. O *Ethanol* não está vinculado a um fornecedor, evitando o aprisionamento tecnológico.

Nos capítulos anteriores mostramos que os conceitos de SDN podem ser utilizados para redes sem fio e que geram benefícios para o funcionamento geral da rede. Implementamos quatro estudos de caso cujos resultados indicam que a arquitetura proposta no capítulo 4 é aplicável ao ambiente de redes locais sem fio. Os estudos de caso apresentados no capítulo 5 mostram que é viável a implementação de uma plataforma SDN para gerenciamento de pontos de acesso sem fio IEEE 802.11 utilizando a arquitetura *Ethanol*.

Os estudos de caso apresentados mostram ser possível realizar o *controle do processo de associação* de estações sem fio aos roteadores *Ethanol*; o *controle de fluxo de dados*, garantindo qualidade de serviço para o usuário; 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; e a *identificação de interfaces sem fio* que não estão transmitindo, utilizando recursos de varredura do protocolo 802.11. Os casos apresentados na dissertação são somente uma amostra do que pode ser construído a partir da arquitetura SDWN proposta. O *Ethanol* pode ser aplicado (programado) para realizar outras funções.

A implementação realizada neste trabalho não permite tirar conclusões de desempenho do funcionamento da arquitetura em um ambiente "real", em função dos testes

realizados terem sido feitos utilizando computadores com placas de rede. Identificamos contudo que os recursos de software necessários estão disponíveis ou podem ser implementados para os dispositivos de rede.

Apresentamos na dissertação uma série de desafios a serem enfrentados no gerenciamento de redes sem fio. Em contraponto a eles, foram mostradas na seção 4.5 como estes desafios podem ser atacados utilizando a arquitetura proposta. Ainda existe muitos aspectos destes desafios a serem explorados pela arquitetura *Ethanol*. Na próxima seção mostramos algumas sugestões para trabalhos futuros com esta plataforma.

6.1 Trabalhos Futuros

Dentre os desafios apresentados na seção 3.2, alguns não foram tratados nos estudos de caso. Consideramos importante a avaliação do modelo no tratamento de *roaming* de usuários, a avaliação de aspectos de segurança e de localização. A arquitetura proposta também permite uma ampla gama de configurações dos parâmetros da rede sem fio que não foram explorados na dissertação. A configuração destes parâmetros permite a elaboração de algoritmos de gerenciamento, inclusive tratando aspectos de diagnóstico remoto, como no caso de uma operadora de banda larga durante o atendimento a um cliente residencial.

Nos estudos de caso, apesar da rede sem fio montada se caracterizar como uma rede densa, ou seja, muitos pontos de acesso em uma pequena área, utilizamos um pequeno número de clientes sem mobilidade. Nossos experimentos não foram realizados em ambientes de alta concentração de usuários, devido à limitação de recursos. Também não foram testados em situações de alta mobilidade e tráfego, que demandassem respostas rápidas do controlador de forma que a latência gerada por este processo de decisão pudesse ser avaliada em situações de sobrecarga. Estes testes devem ser realizados em desenvolvimentos futuros da arquitetura.

Identificamos que diversos protocolos auxiliares ao 802.11, como 802.11k, 802.11r e 802.11v, não estão implementados no ambiente Linux utilizado em nossos experimentos, reduzindo assim o escopo da arquitetura proposta que pôde ser implementado. Estas funções permitem acesso a informações das estações, sem necessidade de instalação de agentes. A implementação total ou parcial destes protocolos poderão ser feitas em trabalhos futuros de modo a permitir ampliar a quantidade de operações que o *Ethanol* pode realizar sobre a rede sem fio.

Em função do tamanho do módulo hostapd, mostrado na seção 4.2.2, e da capacidade dos dispositivos que dispúnhamos, não foi possível avaliar os estudos de caso nos

dispositivos de baixo custo disponíveis. A implementação em dispositivos de mercado com maior capacidade de armazenamento deverá ser realizada em trabalhos futuros. Uma abordagem utilizando comunicação via socket SSL ou pela alteração do protocolo OpenFlow para suportar as mensagens do *Ethanol* pode ser realizado para reduzir, em novos trabalhos, o tamanho final do módulo *hostapd*. Desta forma, pode ser viável a implantação do *Ethanol* em dispositivos com recursos ainda mais limitados.

A implementação realizada no *Ethanol* para manipulação de filas é feita mediante criação de subprocessos que invocam comandos do *OpenvSwitch*. Para realização de uma configuração inicial ou para configurações pouco frequentes que não precisam de desempenho, esta abordagem não apresenta dificuldade de programação nem impacta na execução. Contudo em um ambiente onde a necessidade de configuração é frequente, este método pode não ser recomendado. É possível utilizar a interface *netlink* para manipulação das filas de encaminhamento, utilizando mensagens via *socket* para o *kernel*. A utilização desta interface também evita um problema identificado na implementação atual do *Ethanol* - a dependência de configuração manual da localização dos programas do *OpenvSwitch*.

No estudo de caso de controle de fluxos, utilizamos o escalonador de fila padrão do Linux - o HTB. O *kernel* possui diversos outros escalonadores que podem ser alocados para os fluxos de dados. A arquitetura proposta permite o uso de diversos escalonadores, contudo este aspecto não foi explorado na dissertação.

Versões mais novas do OpenFlow apresentam a possibilidade de utilizar *meter actions* para efetuar a configuração de *QoS*. Esta característica é opcional e não estava implementada na versão do *OpenvSwitch* utilizada nos experimentos. A utilização destas ações constituem uma alternativa à implementação do estudo de caso em 5.2 e pode ser explorada em novos trabalhos.

Nesta dissertação não tratamos dos aspectos de virtualização de redes sem fio. A utilização de conceitos como aqueles apresentados no FlowVisor, OpenVirtex, CloudMac e ODIN pode permitir que os administradores de rede forneçam a rede sem fio como serviço aos seus usuários. A arquitetura do *Ethanol* pode ser explorada em trabalhos futuros para permitir a virtualização da rede sem fio. Por exemplo, com a virtualização pode ser endereçado o problema enfrentado pelas operadoras de banda larga que ao oferecerem a um cliente um roteador com interface de rede sem fio, desejam, também, que seus outros clientes, no raio de alcance deste ponto de acesso sem fio, possam utilizá-lo para acesso à Internet.

A arquitetura do *Ethanol* permite que os parâmetros de configuração e as informações sobre a rede sem fio em cada ponto de acesso controlado possam ser monitorados por um controlador central. Esta característica pode ser explorada para permitir a re-

solução de problemas enfrentados pelas equipes de suporte das operadoras de banda larga que necessitam de acesso remoto às condições da rede de usuário. A utilização do *Ethanol* pode ser utilizada em trabalhos futuros para obter informações sobre como usuários de banda larga utilizam os recursos em suas residências, permitindo à operadora oferecer serviços sob medida, bem como oferecer melhor desempenho e melhor serviço de suporte ao cliente, aumentando o seu lucro e/ou reduzindo o seu custo operacional.

Todos os experimentos realizados durante a dissertação foram realizados com a rede local sem fio configurada como ESS, isto é, a rede sem fio consiste de pontos de acesso no modo infraestrutura e estações sem fio. O protocolo IEEE 802.11 define mais duas arquiteturas - as redes "ad hoc" ou IBSS (*Independent Basic Service Set*) e as redes em malha (IEEE802.11 [2012]). Nas redes em malha os pontos de acesso são interconectados utilizando a própria rede sem fio. Esta configuração gera dificuldades adicionais em uma abordagem SDN, pois agora a comunicação com o controlador sofre os mesmos problemas dos dados de usuário. Além disto, um controlador de uma rede em malha deve tratar ainda das mudanças de topologia pois os nós podem ser móveis. Em função destas alterações, mecanismos de roteamento devem ser implementados. Atualmente estes são mecanismos distribuídos, entretanto em uma abordagem SDN poderíamos explorar protocolos de roteamento baseados no controlador. Os aspectos particulares das redes em malha podem ser estudados em trabalhos futuros.

Com o *Ethanol* fornecemos uma API que permite a criação de algoritmos de controle fechado para o gerenciamento da rede. Esta API permite a obtenção de informações sobre o estado da rede, especificação do comportamento desejado pelo controlador e disseminação para os elementos de rede de novas configurações capazes de prover o comportamento determinado. Muitos algoritmos de controle podem ser explorados em trabalhos futuros, mediante a criação de aplicações que utilizem a API do *Ethanol*.

A implementação atual do *Ethanol* não realiza persistência dos objetos criados no controlador. Esta característica é importante para retornar rapidamente a rede ao seu funcionamento normal no caso da falta de energia no controlador, por exemplo. Existem muitas opções disponíveis para o Python. A implementação de persistência é sugerida para uma futura implementação da plataforma.

A arquitetura *Ethanol* proposta pode ser utilizada em sistema com controladores distribuídos, contudo esta abordagem não foi explorada nesta dissertação. A avaliação do modelo de implementação em controladores distribuídos pode ser estudada em trabalhos futuros.

Apêndice A

Descrição detalhada da API *Ethanol*

A API proposta para o Ethanol utiliza uma abordagem orientada a objetos e é apresentada na figura 4.3 onde as entidades apresentam propriedades, métodos e eventos. As entidades do modelo representam objetos físicos ou virtuais que podem ser configurados ou observados. Por exemplo, um ponto de acesso Ethanol e um fluxo são, respectivamente, uma entidade física e uma entidade virtual.

As entidades apresentadas no modelo possuem propriedades observáveis e/ou configuráveis, tais como o SSID ou o número de clientes associados a um ponto de acesso. As propriedades são acessíveis por meio de métodos GET/SET, que não são apresentados no modelo da figura 4.3 a fim de facilitar a leitura. Os métodos que permitem buscar um ou mais objetos de uma entidade também não são mostrados, como por exemplo o método $getApByMAC(mac)$ da classe $AccessPoint$ - que retorna um ponto de acesso que possui o mac passado como parâmetro - ou $getVAPs()$ da classe $AccessPoint$ - que retorna uma lista com os pontos de acesso virtuais criados na rede.

As entidades também possuem eventos que geram chamadas para o controlador, permitindo que este gere respostas apropriadas a acontecimentos observados, como por exemplo a solicitação de associação de uma estação em um ponto de acesso sem fio - *evUserAssociating()*. Os eventos são identificados pelo prefixo "ev" no nome do método. As propriedades que são somente leitura são identificadas com um sinal de menos ("-").

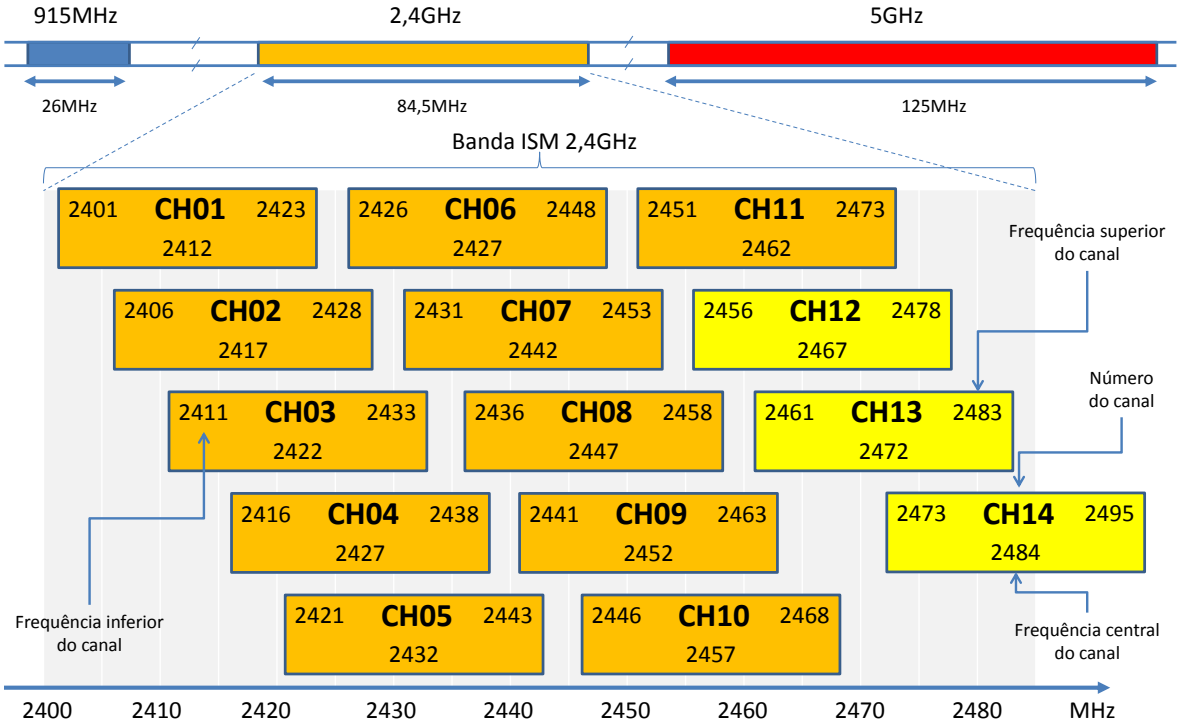
Para cardinalidade foi utilizada a representação de pé de galinha (*Crow's foot notation* (Rob & Coronel [2002])). Um losango preenchido na linha da associação das classes indica contenção (*containment*), que é uma forma de agregação forte, na qual os objetos contidos não possuem existência independente do seu recipiente (*container* que é a classe na qual o losango está conectado).

Nas subseções seguintes explicaremos cada um dos métodos e eventos tratados

por cada objeto do modelo.

A.1 *Radio*

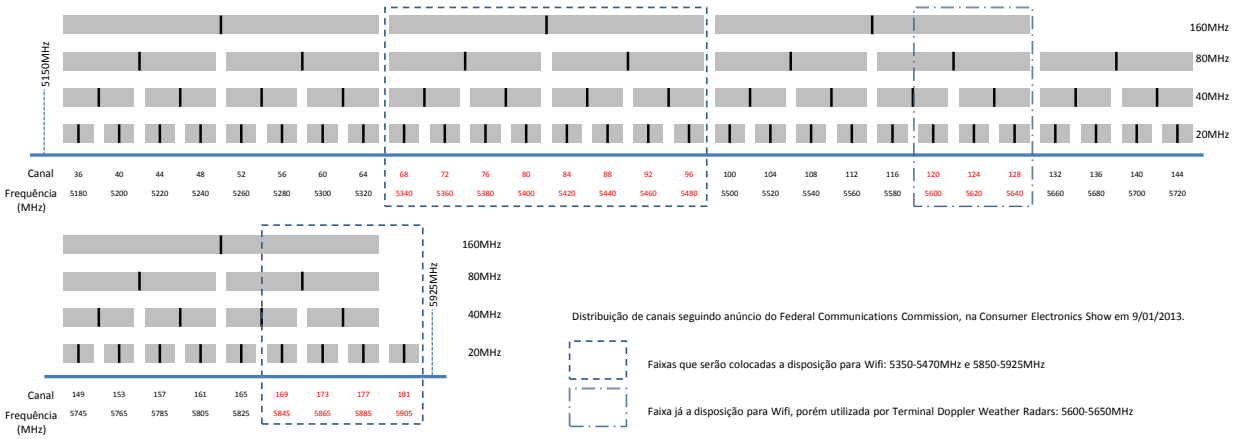
O atributo *currentChannel* representa o canal configurado para transmissão no enlace em questão. Este atributo pode ser acessado utilizando *getCurrentChannel()*. *setCurrentChannel(newChannel)* permite determinar (trocar para) o canal de transmissão. Quando é passado para este método um valor de canal inválido, o canal não é alterado. Similarmente, para o atributo *frequency*, temos *getFrequency()* que retorna a frequência de transmissão do canal selecionado. Cada canal em 802.11 possui uma frequência central, como apresentado para a faixa ISM de 2,4GHz na figura A.1. Na figura A.2 mostramos as frequências centrais para os diversos canais da rede 802.11 em 5GHz. O método *setFrequency(new_frequency)* permite determinar manualmente o canal de transmissão, utilizando para isto o valor da frequência central do canal desejado, sendo que a seleção de uma frequência inválida não efetua qualquer alteração.



Adaptado de [Zheng et al., 2009, p.144] [Coleman & Westcott, 2014, p.222].

Figura A.1: Frequências sobrepostas em 2.4GHz

O método *getChannelBandwidth()* permite obter a largura de canal que está sendo utilizada no enlace (representado pelo atributo *channelBandwidth*), enquanto *setChannelBandwidth(newChannel)* permite determinar a largura de canal a ser utilizada no



Adaptado de `http://community.arubanetworks.com/t5/Technology-Blog/FCC-Announcement-new-5GHz-spectrum-for-Wi-Fi/ba-p/57582`

Figura A.2: Frequências em 5GHz nos Estados Unidos da América

enlace obtendo com isto uma variação na taxa de transmissão. Para 802.11n é possível estabelecer os valores de HT20|HT40+|HT40- (correspondendo a larguras de 20 ou 40MHz), enquanto em IEEE 802.11ac é possível determinar larguras ainda maiores (tabela A.1). Se o parâmetro da nova largura for inválida, nada é alterado. O valor de HT20 corresponde à largura de banda de 20MHz enquanto HT40 corresponde à largura de banda de 40MHz, ou seja, dois canais de 20MHz. Usando HT40-, o canal primário e de controle é o canal superior, enquanto o canal secundário é o inferior. Já na opção HT40+, os canais são invertidos, isto é, o inferior é o canal primário e de controle, enquanto o secundário é o superior.

Número de	**Largura do canal**			
fluxos espaciais	**20MHz**	**40MHz**	**80MHz**	**160MHz**
1	86	200	433	866
2	173	400	866	1,73Gbps
3	288.9	600	1,3Gbps	2,34Gbps
4	346.7	800	1,73Gbps	3,46Gbps

Tabela A.1: Taxas de transmissão em função da largura do canal

O método *getBitrates()* retorna o valor armazenado em *tx_bitrates*. Este método obtém os valores das taxas de bits que serão utilizados pelo ponto de acesso (por padrão, são todos os valores para as taxas de bits suportadas pelo equipamento). O método *setBitrates(new_bitrates)* determina quais os valores das taxas de bits serão efetivamente utilizados pelo equipamento. Se a lista das taxas de bits informada possuir algum valor inválido, o método não efetua alteração na configuração corrente.

A implementação do mac80211 possui uma característica de economia de energia que permite que a interface sem fio seja colocada em um modo de economia de energia dinamicamente, quando ele está associado e não possui tráfego por um certo período de tempo (o valor padrão para este período é de $100ms$). *getPowerSaveMode()* obtém se a opção de economia de energia está ativada ou não. *setPowerSaveMode(new_psm)* determina se o equipamento permitirá o modo de economia de energia se o parâmetro passado foi $True$, alterando o valor do atributo *powerSaveMode*.

O atributo *fragmentationThreshold* pode ser lido pelo método *getFragmentationThreshold()* que retorna o valor configurado para o tamanho máximo dos quadros que serão transmitidos pelo ponto de acesso. Qualquer pacote maior do que o valor definido será fragmentado e enviado em quadros separados. *setFragmentationThreshold(new_threshold)* determina o valor do limiar de fragmentação. O valor padrão dessa condição é 2346 bytes.

O método *getChannelInfo()* retorna informações do canal compreendendo uma lista na qual cada entrada contem: frequência (MHz), ruído (dB), tempo de atividade do canal, tempo de canal ocupado e tempo de transmissão (os valores de tempo são informados em ms).

Podemos obter informações de cada um dos rádios presente no dispositivo usando *getWirelessInterfaceInfo()*, que retorna informações coletadas sobre o hardware do ponto de acesso. Este método obtém diversas informações de configuração bem como modos suportados pela interface.

O método *getLinkStatistics()* corresponde ao relatório fornecido pelo protocolo IEEE 802.11k denominado "*STA statistics*" composto por dois grupos de contadores. O primeiro grupo consiste nos denominados "*STA counters*" que inclui fragmentos transmitidos e recebidos, quadros multicast transmitidos e recebidos, número de falhas, número de tentativas, número de tentativas múltiplas, número de quadros duplicados, transmissão de quadros RTS com sucesso e com falhas, número de falhas de ACK, número de erros de FCS e número de quadros transmitidos e recebidos. O segundo grupo é denominado "*BSS Average Access Delay*" que inclui o atraso médio de acesso ao AP, atraso médio de acesso para cada categoria de acesso, número de estações associadas e utilização do canal.

A.2 AccessPoint

O método *getFastBSSTransition_compatible()* retorna se o ponto de acesso é compatível com o protocolo IEEE 802.11r, que permite transições rápidas entre BSS. Este

valor corresponde ao atributo *fastBSSTransition_compatible* que permite somente leitura. Se esta opção for verdadeira, o protocolo de renegociação da chave de segurança é alterado para que a negociação e a requisição de novos recursos ocorram em paralelo, caso o ponto de acesso e a estação sejam compatíveis.

A informação se as transmissões serão com preâmbulo longo ou curto pode ser obtida chamando *get802_11b_preamble()*. O preâmbulo é um tempo de espera e sincronismo que precede a transmissão de cada quadro, desta forma indica ao receptor que uma transmissão está por vir, permitindo ao receptor sincronizar com o transmissor. Usando o preâmbulo longo (*long*), o tempo de espera é de $192\mu s$, enquanto ao utilizar o preâmbulo curto (*short*) o tempo é reduzido para $96\mu s$, resultando em um pequeno ganho de desempenho. O uso de preâmbulos curtos não é compatível com sistemas legados de rede 802.11 a taxas de 1 e 2Mbps. O valor pode ser alterado de curto para longo e vice versa usando o método *set802_11b_preamble(new_preamble)*. Caso o valor indicado não seja compatível, nenhuma ação será tomada.

É possível determinar se o nível de transmissão do sinal será auto ajustado, se será determinado por um valor fixo ou variará até um limite superior utilizando *setPowerLevelType(new_plt)*. *setPowerLevel(new_pl)* determina o valor da potência de transmissão em dBm. Estes dois métodos definem como se comportará a potência do sinal transmitido.

O intervalo de *beacon* pode ser alterado usando *setBeacon(new_beacon_interval)*. O *beacon* é um quadro enviado periodicamente pelo ponto de acesso, com a função de avisar aos clientes (estações) que a rede está presente, avisar sobre quadros gravados no *buffer* do ponto de acesso aguardando transmissão e também sincronizar a transmissão dos dados. Por padrão, o *beacon* é transmitido a cada $100ms$, mas é possível especificar um valor entre $10ms$ e $1000ms$.

As filas configuradas os portos do comutador OpenFlow[1] são obtidas chamando *getPortQueues()*. Este método retorna um objeto *queues* que contem a informação de todas as configurações existentes no dispositivo e permite a configuração dos parâmetros de *QoS*, ou seja, permite que sejam configuradas filas para cada interface do dispositivo (eth0, eth1, wlan0, etc). Para cada fila é determinado o valor de max_rate e min_rate. *setQueuingDiscipline(new_qdisc)* determina qual disciplina de fila (*qdisc* ou algoritmo de escalonamento) será utilizada[2].

Para criar um AP virtual usamos *createVirtualAP(vap)*. Este método habilita

[1]Utilizando os comandos do *OpenvSwitch* é possível listar as configurações de fila e prioridade dos portos ethernet e portos de rede sem fio.

[2]No Linux com *OpenvSwitch* podemos configurar somente HTB. Outras opções estão disponíveis mediantes chamadas via netlink.

um ponto de acesso vinculado a um SSID em um um ponto de acesso físico. Esta operação está vinculada a um rádio (2,4GHz ou 5GHz). Se a interface suportar mais de um SSID por rádio, a chamada a este método cria os SSIDs desejados. O método *destroyVirtualAP(vap)* remove todas as configurações de SSID do ponto de acesso físico e desabilita esta rede sem fio na interface.

Os modos suportados pelas interfaces sem fio são obtidos usando *getSupportedInterfaceModes()* que retorna uma lista, que poderá conter: IBSS, managed, AP, AP/VLAN, monitor, mesh point, P2P-client, P2P-GO. Este método retorna ainda quais as combinações válidas para os modos retornados. Equivale à informação obtida pela execução do comando *iw list* em ambientes linux (https://wireless.wiki.kernel.org/en/users/Documentation/iw).

Uma visão local dos dispositivos sem fio ao alcance podem ser obtidos por *getInterferenceMap()*, contendo informações como endereço MAC do dispositivo, canal utilizado, potência recebida e SNR (*Signal-to-Noise Ratio*). O método *listWLAN_interfaces()* retorna quais são os nomes das interfaces de rede sem fio no dispositivo físico representado pela classe.

A.3 *Network*

Um objeto desta classe agrupa todos os pontos de acesso virtuais em uma rede local sem fio que compartilham o mesmo valor para o SSID e que estão associados ao controlador Ethanol.

O SSID configurado para a rede (objeto desta classe) pode ser obtido usando *getSSID()*. *setSSID(new_ssid)* define o nome do SSID a ser atribuído à rede. Se a rede já estiver ativa, corresponde a alterar o SSID atribuído a todos os pontos de acesso virtuais que participam da rede.

O método *associateVirtualAP(vap)* realiza a associação de um AP virtual a uma rede. Este procedimento corresponde à criação de um ponto de acesso virtual (vinculado a um ponto de acesso físico), isto é, a criação e configuração do SSID da rede no ponto de acesso físico e sua habilitação. *deassociateVirtualAP(vap)* desassocia um AP virtual da rede, removendo este ponto de acesso virtual da rede (SSID) do ponto de acesso físico e invocando *VirtualAccessPoint.setEnabled(False)*.

O método *handoffUser(station,new_ap)* realiza o handoff de um usuário (estação) transferindo a estação para um novo ponto de acesso e retirando-o do ponto de acesso inicial. O algoritmo de seleção do AP de destino deve ser feita por um procedimento, não definido na API do Ethanol. O método de *handoff* recebe o AP de origem e de

destino como parâmetro, bem como o usuário que será migrado.

A.4 *Station*

Os endereços MAC, IP versão 4 e versão 6 da estação podem ser obtidos, respectivamente, chamando *getMac_ address()*, *getIpv4_ address()* e *getIpv6_ address()*. Estes valores podem ser definidos, com a respectiva máscara e gateway pelos métodos *setIpv4_ address(new_ address[,mask[,gw]])* e *setIpv6_ address(new_ address[,mask[,gw]])*. O endereço de camada de enlace poderá ser alterado utilizando *setMac_ address*, se a interface suportar este recurso.

Para determinar se a estação é compatível com o protocolo 802.11e ou não, chamamos *get802_ 11e_ enabled()*. Já *getFastBSSTransition_ compatible()* retorna se o ponto de acesso é compatível com o protocolo IEEE 802.11r, que permite transições rápidas entre BSS. Utilizando *triggerTransition*, o controlador pode enviar à estação, que implementa o protocolo IEEE 802.11v, uma solicitação para que este troque sua conexão para um ponto de acesso determinado.

Os valores das taxas de bits que serão utilizados pelo ponto de acesso (por default são todos os valores para as taxas de bits suportadas pelo equipamento) podem ser obtidos usando *getBitrates()*. A chamada para *setBitrates(new_ bitrates)* determina quais os valores das taxas de bits que serão efetivamente utilizados pelo equipamento (um subconjunto não vazio de *getBitrates()*), alterando o valor do atributo *tx_ bitrate*.

Os métodos *getBytesReceived*, *getPacketsReceived()*, *getBytesSent()*, *getPacketsSent()*, *getBytesLost()* e *getPacketsLost()* permitem obter os valores acumulados dos bytes e pacotes recebidos, transmitidos e perdidos, nesta ordem.

No contexto das redes de computadores, *jitter* é a variação na latência, medida pela variabilidade ao longo do tempo da latência de pacotes em uma rede, desta forma uma rede com latência constante não tem variação, ou seja, tem *jitter* nulo. A API retorna esta medida utilizando *getJitter()* e retorna o atraso de transmissão usando *getDelay()*. *getSignalStrength()* retorna a intensidade instantânea do sinal enquanto *getSNR()* obtém o valor instantâneo da relação sinal/ruído medido pelo último quadro recebido.

O tempo da conexão da estação ao ponto de acesso pode ser recuperado com *getUptime()*. O número de tentativas de transmissão realizadas pela estação é retornado por *getRetries()* e o número de tentativas de transmissão que falharam é retornado por *getFailed()*.

Uma visão local dos dispositivos sem fio ao alcance podem ser obtidos por *ge-*

tInterferenceMap(), contendo informações como endereço MAC do dispositivo na vizinhança da estação, o canal utilizado por este dispositivo, a potência recebida e o SNR.

Uma variação do "Neighbor report" definido no IEEE 802.11k é gerada por *getAPsInRange()* que retorna uma lista contendo todos os pontos de acesso detectados pela estação (scan passivo). Cada entrada contem: endereço MAC do AP, a interface (caso exista mais de uma interface sem fio), se está associado ou não a um AP, a frequência, a intensidade do sinal (em dBm) e o SSID. Parte das informações são retiradas da entrada da MIB denominada dot11RRMNeighborReportTable para cada AP. O relatório do 802.11k retorna os APs que são candidatos a uma transição de conjunto de serviço.

O método *getChannelInfo()* obtém informação do canal que está conectado contendo as informações o endereço MAC do BSS (AP), se está associado ou não, o valor do TSF (*Timing Synchronization Function*), a frequência central do canal, o intervalo do *beacon*, o intervalo do último *beacon* recebido, o valor do sinal em dBm, SSID e as taxas de bits suportadas.

O método *getBeaconInfo()* equivale ao "*Beacon Report*" definido no protocolo 802.11k, que obtém o conteúdo corrente de beacons armazenados na estação para todos os canais suportados pelo SSID e BSSID, sem realizar medições adicionais.

O método *getNoiseInfo()* equivale ao relatório "*Noise Histogram*" do protocolo IEEE 802.11k. Nossa implementação contém menos informações, compreendendo uma lista onde cada entrada contém: frequência (MHz), ruído (dB) e SNR.

O método *getLinkMeasurement()* permite obter as características de radiofrequência da estação para o enlace. Este medida, como definida pelo protocolo 802.11k, busca indicar o qualidade instantânea do enlace.

Os resultados retornados por *getStatistics()* corresponde ao relatório fornecido pelo protocolo IEEE 802.11k denominado "*STA statistics*" composto por dois grupos de contadores. O primeiro grupo consiste nos denominados "*STA counters*" que inclui fragmentos transmitidos e recebidos, quadros multicast transmitidos e recebidos, número de falhas, número de tentativas, número de tentativas múltiplas, número de quadros duplicados, transmissão de quadros RTS com sucesso e com falhas, número de falhas de ACK, número de erros de FCS e número de quadros transmitidos e recebidos. O segundo grupo é denominado "*BSS Average Access Delay*" que inclui o atraso médio de acesso ao AP, atraso médio de acesso para cada categoria de acesso, número de estações associadas e utilização do canal.

Uma estação que implementa IEEE 802.11v e possui informação de localização (como acesso a dados de GPS) pode ter sua localização solicitada pelo controlador

mediante chamada à *getLocation*.

A.5 *VirtualAccessPoint*

O método *getEnabled()* verifica se o AP virtual está habilitado ou não. Fazendo *setEnabled(true|false)* igual a $True$, o AP virtual será habilitado. Se for desabilitado, as configurações permanecem, porém o SSID fica desativado (não transmite ou recebe quadros e não pode ser detectado por estações sem fio).

O método *getBroadcastSSID()* retorna se o SSID está sendo transmitido no beacon ou não. Já *setBroadcastSSID(true|false)* permite habilitar se o SSID será transmitido no beacon, ao ser igualado à $True$, ou desabilita esta transmissão, se o valor for $False$. Desabilitar a transmissão do SSID equivale a colocar o ponto de acesso como "invisível", isto é, ele ainda continua transmitindo mas não aparece nas listas de redes sem fio disponíveis.

O método *getFastBSSTransition_compatible()* retorna se o ponto de acesso é compatível com o protocolo IEEE 802.11r, que permite transições rápidas entre BSS.

As informações de segurança configuradas são recuperadas por *getSecurity()*, indicando se a rede é aberta, WEP, $WPA-PSK$ ou WPA (versão 1 ou 2) e suas configurações específicas, tais como chaves, encriptação, informações de RADIUS (IEEE 802.11x). As configurações podem ser alteradas utilizando *setSecurity(new_security)*, utilizando os parâmetros adequados.

O método *getContention()* obtém os valores de contenção da rede sem fio, que correspondem aos valores do AIFS, cwMax e cwMin (valores máximo e mínimo para as janelas de contenção). *setContention(new_contention)* permite definir os valores para cada um destes três parâmetros de contenção. Se o ponto de acesso for compatível com IEEE 802.11e estes parâmetros poderão ser alterados para cada uma das classes de acesso suportadas.

O método *getCAC()* obtém informações de *Call Admission Control* (utilizada em transmissões de voz em redes sem fio) composta pelas seguintes parâmetros: se está habilitado ou não (*callAdmissionControlEnabled*); qual a vazão máxima para tráfego de voz (*maxVoiceTrafficBandwidth*); e qual a vazão reservada para cliente roaming (*bandwidthReservedRoamingClients*). Já *setCAC(new_cac)* permite atribuir os valores para estes três parâmetros. Estes métodos dependem da identificação automática do tráfego de voz.

A transmissão de quadros em rajadas (*frame burst*) é uma técnica utilizada em protocolos de comunicação para mídias compartilhadas, como as redes sem fio, bus-

cando maior desempenho ao permitir que o transmissor envie uma série de quadros em sucessão, sem abrir mão do controle do meio de transmissão (veja figura A.3). A especificação do protocolo IEEE 802.11e define como se dá a transmissão em rajadas em comunicações entre ponto de acesso e estações (e vice-versa) e entre clientes em uma rede ad hoc. *getFrameBurst()* retorna se a transmissão por rajadas está habilitada ou não. Já *setFrameBurst(true/false)* permite habilitar (*True*) ou desabilitar (*False*) este recurso.

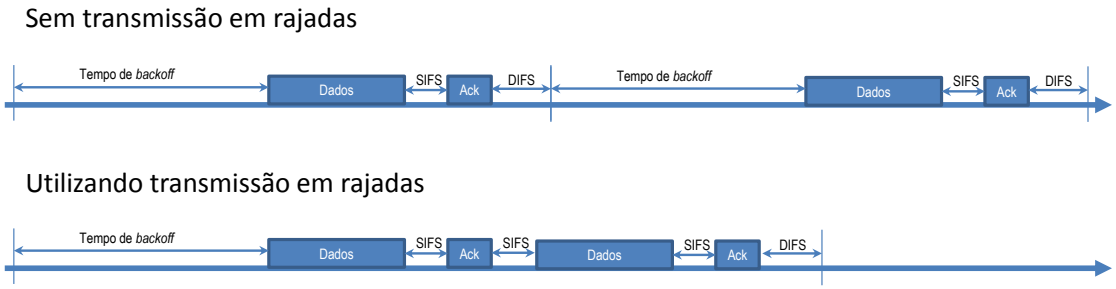


Figura A.3: Esquema comparativo entre transmissões por rajadas ou sem rajadas
Adaptado de Kolap [2012].

O período definido pelo intervalo de guarda é o tempo criando entre símbolos que são transmitidos na rede sem fio. Este intervalo de guarda existe para tentar eliminar a interferência entre símbolos transmitidos - ISI (*Intersymbol Interference*). Esta interferência acontece quando ecos ou reflexões de um símbolo interferem com a transmissão de outro símbolo. Aumentar o período entre as transmissões dos símbolos permite que esses ecos e reflexões sejam atenuados ou tratados antes que o próximo símbolo seja transmitido. O método *getGuardInterval()* retorna o tempo que está configurado. Com 802.11n, os intervalos de guarda podem ser alterados. Para isto podemos utilizar *setGuardInterval(new_guard_interval)* alterar o valor deste parâmetro. Caso não seja possível, nada é reportado. O intervalo de guarda padrão utilizado no protocolo IEEE 802.11 para OFDM é de $0,8\mu s$.

DTIM indica o número de *beacons* que o ponto de acesso aguarda antes de transmitir pacotes de *multicast* agendados e portanto o intervalo entre DTIMs efeta as transmissões de pacotes *multicast* (transmitidos simultaneamente a várias estações). *getDTIMInterval()* retorna o valor definido para este parâmetro, enquanto *setDTIMInterval(new_dtim)* define um novo valor. Os valores aceitos estão entre 1 e 255, sendo que o padrão é 3. Quanto maior é o valor, menor é a prioridade dos pacotes de *multicast*.

O padrão IEEE 802.11 implementa um controle de colisões, denominado RTS/CTS, que consiste em um processo de verificação mediante troca de mensagens,

onde o cliente envia um quadro RTS (*Request to Send*) e aguarda o recebimento de um quadro CTS (*Clear to Send*) antes de começar a transmitir. O quadro CTS funciona como uma autorização para transmitir enviada pelo receptor ao transmissor, avisando às demais estações que uma transmissão está prestes a ser iniciada e que qualquer transmissão das estações ao alcance do receptor deve ser adiada. Como todas as estações mantêm contato com o ponto de acesso, estas recebem quadros CTS enviados por ele e sabem que devem esperar sua vez antes de transmitir qualquer coisa. O método *getRTSThreshold()* retorna o valor do limiar do RTS configurado no ponto de acesso virtual, enquanto *setRTSThreshold(new_threshold)* permite definir o valor deste limiar. Os valores aceitos estão na faixa de 256 a 2432, sendo 2432 o valor padrão deste parâmetro. *getCTSProtection_enabled()* retorna se o controle mediante envio de RTS/CTS está habilitado, enquanto *setCTSProtection_enabled(true|false)* habilita ou desabilita este controle.

Uma estação pode ser forçada a se desassociar de um ponto de acesso usando o método *disassociateUser(station)*, que remove da tabela de associações interna ao ponto de acesso às informações da estação que desejamos desassociar, e, também, pode ter suas informações de autenticação removidas usando *deauthenticateUser(station)*, forçando que a estação reautentique para poder continuar a transmitir por aquele ponto de acesso.

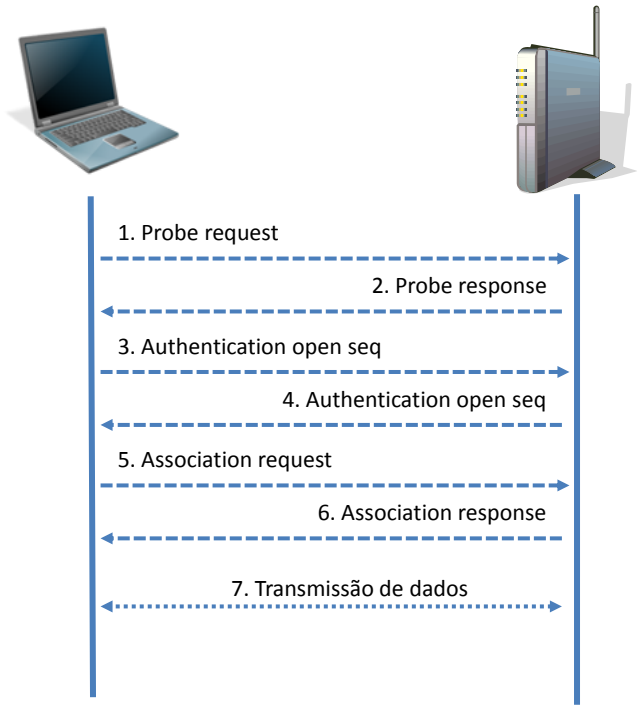


Figura A.4: Processo de associação de uma estação à rede sem fio

O processo de associação de uma estação a um BSS pode ser sumarizado pelas etapas mostradas na Figura A.4. Uma estação e um AP trocam uma série de quadros de gerenciamento definidos pelo protocolo 802.11, a fim de chegar a um estado autenticado e associado. A estação deve estar autenticada e associada antes poder transmitir dados. Na API do Ethanol um evento é gerado para o inicio da associação (*evUserConnecting*), quando a resposta da *probe* é recebida (*evProbeRequestReceived*), quando a estação solicita autenticação (*evUserAuthenticating*), quando a estação solicita a associação (*evUserAssociating*). Um cliente utiliza quadros de reassociação para se associar a um novo AP. O processo de reassociação gera um evento denominado *evUserReassociating* e, quando a transição rápida é possível, gera *evFastReassociation*. Durante a transição de um AP para o seguinte, é gerado um evento denominado *evFastTransition*.

O processo de desconexão de uma estação também gera eventos quando a estação solicita desconexão (*evUserDisconnecting*) e quando está desassociando (*evUserDisassociating*). Todos estes eventos são tratados no controlador que pode informar ao ponto de acesso se o processo deve continuar ou deve ser abortado.

Os métodos *registerMgmtFrame(mgmt_frame, handler)* e *unregisterMgmtFrame(mgmt_frame)* permitem definir quais as mensagens serão encaminhadas ao controlador e quais serão tratadas no ponto de acesso usando o procedimento normal definido pelo IEEE 802.11. Quando as mensagens de gerenciamento cadastradas são recebidas pelo ponto de acesso, o evento $evMgmtFrameReceived$ é criado, simulando um comportamento similar ao CAPWAP (Calhoun et al. [2009]). Assim as mensagens de gerenciamento da rede sem fio são enviadas ao controlador.

A.6 *Flow*

O método *createFlow()* permite criar um objeto capaz de tratar um fluxo especificado pelos *match fields* definidos pelo OpenFlow ([Foundation, 2013, p.55]). Este método somente cria o objeto de acordo com o modelo do *Ethanol*, porém para que as regras de fluxo sejam gravadas no comutador OpenFlow é necessário chamar *apply()*.

Ao criar um fluxo de dados, a ele é atribuído um tempo máximo antes que o fluxo seja descartado (*hardTimeout*) e o tempo que o fluxo pode ficar sem transmitir antes que seja removida da tabela de fluxos (*softTimeout*[3]). Estes dois parâmetros podem ser obtidos, respectivamente, por *getHardTimeout()* e *getSoftTimetout()*. Estes valores são lidos da estrutura de dados ofp_flow_mod do OpenFlow.

[3]O OpenFlow denomina este parâmetro *Idle Timeout*.

Uma classe de serviço (CoS) é uma forma de gerir o tráfego em uma rede por grupos de tipos semelhantes de tráfego em conjunto e tratar cada tipo como uma classe com o seu próprio nível de prioridade de serviço. Podemos obter as informações de CoS configuradas para um determinado fluxo usando *getCOS* ou alterar o valor do CoS usando *setCOS*. A alteração do CoS para um fluxo pode implicar na realocação deste fluxo para uma outra fila ou na configuração dos parâmetros de fila do comutador OpenFlow.

O método *setQueue(queue_id)* corresponde à mensagem OFPAT_SET_QUEUE do OpenFlow, atribuindo o fluxo de dados a uma fila do comutador. A fila a que o fluxo foi atribuído pode ser obtida utilizando *getQueue()*.

Quando o controlador OpenFlow recebe o primeiro pacote de um fluxo, é gerado o evento *evPacketIn* que pode ser tratado por um procedimento de usuário. Caso este procedimento de usuário não esteja definido, será realizado um tratamento padrão que determina ser todos os pacotes deste fluxo devem ser descartados ou modificados utilizando os métodos *dropPacket()* e *modifyPacket()*, caso definidos.

A coleta de estatísticas por um controlador OpenFlow é realizada mediante um processo de troca de mensagens, no qual o controlador solicita que o comutador envie estatísticas de fluxo por porto, pelas entradas de fluxo e pelas tabelas de fluxo, e em contrapartida o comutador responde a estas solicitações com uma mensagem contendo os dados demandados ou erro, caso não tenha suporte para esta característica. Na API do *Ethanol*, dois métodos são responsável por encapsular estas demandas. *getFlowStatistics()* solicita ao comutador as estatísticas do fluxo, isto é, programa o comutador para enviar os dados desejados. Ao receber a resposta do comutador, é ativado no controlador o evento *evFlowStatisticsReceived* que pode ser tratado por um procedimento de usuário, caso definido.

Os métodos *getNumPackets()* e *getNumBytes()* retornam, respectivamente, o número de pacotes e bytes transmitidos pelo fluxo desde a sua criação. *getTimeElapsed()* retorna o tempo decorrido desde a criação do fluxo (em ms). A variação na latência medida para um determinado fluxo de dados é obtida na API utilizando *getJitter()*. Já o atraso de transmissão é obtido usando *getDelay()*.

Referências Bibliográficas

Aggarwal, B.; Bhagwan, R.; Das, T.; Eswaran, S.; Padmanabhan, V. N. & Voelker, G. M. (2009). Netprints: Diagnosing home network misconfigurations using shared knowledge. Em *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'09, pp. 349--364, Berkeley, CA, USA. USENIX Association.

Al-Shabibi, A.; De Leenheer, M.; Gerola, M.; Koshibe, A.; Parulkar, G.; Salvadori, E. & Snow, B. (2014a). Openvirtex: Make your virtual SDNs programmable. Em *ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pp. 25--30.

Al-Shabibi, A.; De Leenheer, M.; Gerola, M.; Koshibe, A.; Snow, W. & Parulkar, G. (2014b). Openvirtex: A network hypervisor. *Open Networking Summit*.

Ali-Ahmad, H.; Cicconetti, C.; De La Oliva, A.; Mancuso, V.; Reddy Sama, M.; Seite, P. & Shanmugalingam, S. (2013). An sdn-based network architecture for extremely dense wireless networks. Em *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pp. 1–7.

Anderson, T.; Peterson, L.; Shenker, S. & Turner, J. (2005). Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41. ISSN 0018-9162.

Aruba Networks, I. (2009). Aruba technical brief:aruba networks position statement on capwap. Accessed: 2015-04-15.

Aruba Networks, I. (2013). Aruba whitepaper:the indestructible network: Wireless lans for industrial and outdoor applications. Accessed: 2015-04-15.

Aruba Networks, I. (2014). Aruba whitepaper: Deploying aruba wireless controllers with microsoft lync SDN API. Accessed: 2015-04-15.

Aruba Networks, I. (2015). Aruba controllers. `http://www.arubanetworks.com/products/networking/controllers/`. Accessed: 2015-05-05.

Azodolmolky, S.; Wieder, P. & Yahyapour, R. (2013). Cloud computing networking: challenges and opportunities for innovations. *Communications Magazine, IEEE*, 51(7):54–62. ISSN 0163-6804.

Bahl, P.; Chandra, R.; Padhye, J.; Ravindranath, L.; Singh, M.; Wolman, A. & Zill, B. (2006). Enhancing the security of corporate wi-fi networks using DAIR. Em *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*, MobiSys '06, pp. 1--14, New York, NY, USA. ACM.

Bansal, M.; Mehlman, J.; Katti, S. & Levis, P. (2012). OpenRadio: A programmable wireless dataplane. Em *HotSDN '12 Proceedings of the first workshop on Hot topics in software defined networks*, pp. 109–114.

Berman, M.; Elliott, C. & Landweber, L. (2014). Geni: Large-scale distributed infrastructure for networking and distributed systems research. Em *Communications and Electronics (ICCE), 2014 IEEE Fifth International Conference on*, pp. 156--161. IEEE.

Betts, M.; Davis, N.; Dolin, R.; Doolan, P.; Fratini, S.; Hood, D.; Joshi, M.; Lam, K.; Mack-Crane, B.; Mansfield, S.; Menezes, P.; Natarajan, S.; Paul, M.; Pourzandi, M.; Rexford, J.; Sadler, J.; Schaller, S.; Schneider, F.; Shefer, S.; Smith, P.; Tourhilles, J.; Tsou, T.; Varma, E.; Vissers, M.; Woolward, M. & Dacheng, Z. (2014). SDN Architecture Overview. Relatório técnico, Open Network Foundation.

Bhaumik, S.; Chandrabose, S. P.; Jataprolu, M. K.; Kumar, G.; Muralidhar, A.; Polakos, P.; Srinivasan, V. & Woo, T. (2012). CloudIQ: A framework for processing base stations in a data center. Em *International Conference on Mobile Computing and Networking (MobiCom)*, pp. 125--136.

Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G. & Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87--95. ISSN 0146-4833.

Bosshart, P.; Gibb, G.; Kim, H.-S.; Varghese, G.; McKeown, N.; Izzard, M.; Mujica, F. & Horowitz, M. (2013). Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. Em *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pp. 99--110.

Cai, Z.; L. Cox, A. & Ng, T. S. E. (2013). Maestro: A system for scalable openflow control.

Calhoun, P.; Montemurro, M. & Stanley, D. (2009). Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification. RFC 5415 (Proposed Standard).

Calvert, K. L.; Edwards, W. K.; Feamster, N.; Grinter, R. E.; Deng, Y. & Zhou, X. (2010). Instrumenting home networks. Em *HomeNets '10 Proceedings of the 2010 ACM SIGCOMM workshop on Home networks*, pp. 55–60.

Capgemini Consulting, G. (2009). Quest for margins: Operational cost strategies for mobile operators in europe. `https://www.capgemini-consulting.com/resource-file-access/resource/pdf/tl_Operational_Cost_Strategies_for_Mobile_Operators_in_Europe.pdf`. Accessed: 2015-05-14.

Casado, M.; Freedman, M.; Pettit, J.; Luo, J.; Gude, N.; McKeown, N. & Shenker, S. (2009). Rethinking enterprise network control. *Networking, IEEE/ACM Transactions on*, 17(4):1270–1283. ISSN 1063-6692.

Casado, M.; Freedman, M. J.; Pettit, J.; Luo, J.; McKeown, N. & Shenker, S. (2007). Ethane: taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1--12. ISSN 0146-4833.

Casado, M.; Koponen, T.; Ramanathan, R. & Shenker, S. (2010). Virtualizing the network forwarding plane. Em *PRESTO '10 Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*.

Casado, M.; Koponen, T.; Shenker, S. & Tootoonchian, A. (2012). FABRIC: A retrospective on evolving SDN. Em *HotSDN 12*.

Chaudet, C. & Haddad, Y. (2013). Wireless software defined networks: Challenges and opportunities. Em *Microwaves, Communications, Antennas and Electronics Systems (COMCAS), 2013 IEEE International Conference on*, pp. 1–5.

Chen, J.-L.; Ma, Y.-W.; Kuo, H.-Y. & Hung, W.-C. (2014). Enterprise visor: A software-defined enterprise network resource management engine. Em *System Integration (SII), 2014 IEEE/SICE International Symposium on*, pp. 381--384. IEEE.

Cheng, Y.-C.; Bellardo, J.; Benkö, P.; Snoeren, A. C.; Voelker, G. M. & Savage, S. (2006). Jigsaw: Solving the puzzle of enterprise 802.11 analysis. Em *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pp. 39--50, New York, NY, USA. ACM.

Chetty, M.; Sung, J.-Y. & Grinter, R. E. (2007). How smart homes learn: The evolution of the networked home and household. Em *UbiComp '07 Proceedings of the 9th international conference on Ubiquitous computing*, pp. 127–144.

Chintalapudi, K.; Padmanabha Iyer, A. & Padmanabhan, V. N. (2010). Indoor localization without the pain. Em *ACM International Conference on Mobile Computing and Networking (MobiCom)*, pp. 173--184.

Choi, S.; del Prado, J.; Sai Shankar, N. & Mangold, S. (2003). IEEE 802.11 e contention-based channel access (EDCF) performance evaluation. Em *IEEE International Conference on Communications (ICC)*, pp. 1151–1156 vol.2.

Cisco Systems, I. (2010). *Cisco Wireless LAN Controller Configuration Guide*. Cisco Systems, Inc, San Jose, CA, USA, 9034729-06 edição.

Cisco Systems, I. (2015a). Cisco wireless lan controllers. `http://www.cisco.com/c/en/us/products/wireless/wireless-lan-controller/index.html`. Accessed: 2015-05-05.

Cisco Systems, I. (2015b). Meraki whitepaper:network topology. Accessed: 2015-04-15.

Coleman, D. D. & Westcott, D. A. (2014). *CWNA Certified Wireless Network Administrator Official Study Guide (Exam PW0-100), Fourth Edition*. John Wiley and Sons, Inc., Indianapolis, Indiana, USA, 4 edição. ISBN 9781118893708.

Cormode, G. & Muthukrishnan, S. (2005). An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58--75.

Dell (2015). Dell w-series controller-based platforms. `http://www.dell.com/us/business/p/powerconnect-w-controllers/pd`. Accessed: 2015-05-05.

Dely, P. (2012). Architectures and algorithms for future wireless local area networks. Accessed: 2015-04-15.

Dely, P.; Vestin, J.; Kassler, A.; Bayer, N.; Einsiedler, H. & Peylo, C. (2012). CloudMAC - an openflow based architecture for 802.11 MAC layer processing in the cloud. Em *Globecom Workshops (GC Wkshps), 2012 IEEE*, pp. 186–191.

Devera, M. (2002). HTB linux queuing discipline manual - user guide. Accessed: 2015-04-15.

DiCioccio, L.; Teixeira, R. & Rosenberg, C. (2011). Characterizing home networks with homenet profiler. *UPMC Sorbonne Universits, Tech. Rep. CP-PRL-2011-09-0001*.

Doo, K.-H.; Yoon, B.-Y.; Lee, B.-C.; Lee, S.-S.; Han, M.-S. & Kim, W.-W. (2012). Multicore flow processor with wire-speed flow admission control. *ETRI Journal*, 34(6):827–837. ISSN 1063-6692.

Duan, Q.; Yan, Y. & Vasilakos, A. (2012). A survey on service-oriented network virtualization toward convergence of networking and cloud computing. *Network and Service Management, IEEE Transactions on*, 9(4):373–392. ISSN 1932-4537.

Erickson, D. (2013). The beacon openflow controller. Em *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pp. 13--18, New York, NY, USA.

Extreme Networks, I. (2014). *IdentiFi^{TM} Wireless User Guide, v9.12.XX*. Extreme Networks, Inc, San Jose, CA, USA, ol-21524-01 edição.

Extreme Networks, I. (2015). Extreme identifi wireless. `http://www.extremenetworks.com/products/wireless`. Accessed: 2015-05-05.

Farrel, A. & Bryskin, I. (2005). *GMPLS: Architecture and Applications (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 0120884224.

Feamster, N. (2010). Outsourcing home network security. Em *Proceedings of the 2010 ACM SIGCOMM Workshop on Home Networks*, HomeNets '10, pp. 37--42.

Feamster, N.; Rexford, J. & Zegura, E. (2013). The road to SDN: An intellectual history of programmable networks. *Queue - Large-Scale Implementations*, 11(12):20–32.

Fernandez, M. (2013). Evaluating openflow controller paradigms. Em *ICN 2013, The Twelfth International Conference on Networks*, pp. 151--157.

Foundation, T. O. N. (2012). Software-Defined Networking: The new norm for networks. Relatório técnico, The Open Networking Foundation.

Foundation, T. O. N. (2013). Openflow switch specification version 1.4.0. Relatório técnico, The Open Networking Foundation.

Gast, M. (2005). *802.11 wireless networks: the definitive guide*. O'Reilly Media, Inc.

Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N. & Shenker, S. (2008). NOX: Towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110.

Gudipati, A.; Perry, D.; Li, L. E. & Katti, S. (2013). SoftRAN: Software defined radio access network. Em *HotSDN '13*.

Guedes, D.; Vieira, L. F. M.; Vieira, M. M.; Rodrigues, H. & Nunes, R. V. (2012). Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. Em *Minicursos do XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos-SBRC*.

Haleplidis, E.; Pentikousis, K.; Denazis, S.; Salim, J. H.; Meyer, D. & Koufopavlou, O. (2015). Software-defined networking (SDN): Layers and architecture terminology. RFC 7426 (Informational).

Heller, B.; Seetharaman, S.; Mahadevan, P.; Yiakoumis, Y.; Sharma, P.; Banerjee, S. & McKeown, N. (2010). Elastictree: Saving energy in data center networks. Em *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, p. 16.

Hiertz, G.; Denteneer, D.; Stibor, L.; Zang, Y.; Costa, X. & Walke, B. (2010). The ieee 802.11 universe. *Communications Magazine, IEEE*, 48(1):62–70. ISSN 0163-6804.

HP Networks, I. (2015). Hp msm controller series. `http://www8.hp.com/us/en/products/networking-wireless/product-detail.html?oid=3963981`. Accessed: 2015-05-05.

Hu, F.; Hao, Q. & Bao, K. (2014). A survey on software-defined network and openflow: From concept to implementation. *Communications Surveys Tutorials, IEEE*, 16(4):2181–2206. ISSN 1553-877X.

Huang, P.-J.; Tseng, Y.-C. & Tsai, K.-C. (2006). A fast handoff mechanism for ieee 802.11 and iapp networks. Em *Vehicular Technology Conference, 2006. VTC 2006-Spring. IEEE 63rd*, volume 2, pp. 966–970. ISSN 1550-2252.

IEEE802.11 (2012). Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2012*, p. 2695p.

IEEE802.11aa (2012). Local and metropolitan area networks specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 2: Mac enhancements for robust audio video streaming. *IEEE Std 802.11aa-2012*, p. 162p.

IEEE802.11ac (2013). Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications–amendment 4: Enhancements for very high throughput for operation in bands below 6 ghz. *IEEE Std 802.11ac-2013*, p. 425p.

IEEE802.11ad (2012). Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 3: Enhancements for very high throughput in the 60 ghz band. *IEEE Std 802.11ad-2012*, p. 628p.

IEEE802.11ae (2012). Local and metropolitan area networks specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications - amendment 1: Prioritization of management frames. *IEEE Std 802.11ae-2012*, p. 52p.

IEEE802.11e (2005). Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications: Amendment 8: Medium access control (MAC) quality of service enhancements. *IEEE Std 802.11e-2005*, p. 189p.

IEEE802.11f (2003). Ieee trial-use recommended practice for multi-vendor access point interoperability via an inter-access point protocol across distribution systems supporting ieee 802.11 operation. *IEEE Std 802.11f-2003*, p. 79p.

IEEE802.11k (2008). Local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac)and physical layer (phy) specifications amendment 1: Radio resource measurement of wireless lans. *IEEE Std 802.11k-2008*, p. 244p.

IEEE802.11r (2008). Local and metropolitan area networks– specific requirements– part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 2: Fast basic service set (bss) transition. *IEEE Std 802.11r-2008*, p. 126p.

IEEE802.11v (2011). Local and metropolitan area networks - specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications - amendment 8: Ieee 802.11 wireless network management. *IEEE Std 802.11v-2011*, p. 433p.

IEEE802.21 (2009). Local and metropolitan area networks part 21: Media independent handover services. *IEEE Std 802.21-2008*, p. 323p.

Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; Zolla, J.; Hölzle, U.; Stuart, S. & Vahdat, A.

(2013). B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3--14. ISSN 0146-4833.

Jammal, M.; Singh, T.; Shami, A.; Asal, R. & Li, Y. (2014). Software-defined networking: State of the art and research challenges. *CoRR*, abs/1406.0124.

Jin, X.; Li, L. E.; Vanbever, L. & Rexford, J. (2013). SoftCell: Scalable and flexible cellular core network architecture. Em *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pp. 163--174.

Juniper Networks, I. (2014). *Mobility System Software, Command Reference Guide*. Juniper Networks, Inc, Sunnyvale, CA, USA.

Kim, H.; Sundaresan, S.; Chetty, M.; Feamster, N. & Edwards, W. K. (2011). Communicating with CAPS: Managing usage CAPS in home networks. Em *ACM SIGCOMM 11(Demo session)*, pp. 115–120.

Kolap, J. (2012). Frame aggregation mechanism for high-throughput 802.11n wlans. *International Journal of Wireless and Mobile Networks - IJWMN*, 30(4):141–153. ISSN 0975-4679.

Kolias, C.; Ahlawat, S.; Ashton, C.; Cohn, M.; Manning, S. & Nathan, S. (2012). Openflow-enabled mobile and wireless networks. Relatório técnico, The Open Networking Foundation. Accessed: 2015-04-15.

Koponen, T.; Casado, M.; Gude, N.; Stribling, J.; Poutievski, L.; Zhu, M.; Ramanathan, R.; Iwata, Y.; Inoue, H.; Hama, T. & Shenker, S. (2010). ONIX: a distributed control platform for large-scale production networks. Em *OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation*.

Kreutz, D.; Ramos, F. M. V.; Veríssimo, P.; Rothenberg, C. E.; Azodolmolky, S. & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *CoRR*, abs/1406.0440.

Kumar, H.; Gharakheili, H. & Sivaraman, V. (2013a). User control of quality of experience in home networks using sdn. Em *Advanced Networks and Telecommuncations Systems (ANTS), 2013 IEEE International Conference on*, pp. 1–6. ISSN 2153-1676.

Kumar, S.; Cifuentes, D.; Gollakota, S. & Katabi, D. (2013b). Bringing cross-layer mimo to today's wireless lans. Em *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pp. 387--398, New York, NY, USA. ACM.

Levin, D.; Canini, M.; Schmid, S. & Feldmann, A. (2013). Incremental sdn deployment in enterprise networks. *SIGCOMM Comput. Commun. Rev.*, 43(4):473--474. ISSN 0146-4833.

Levin, D.; Canini, M.; Schmid, S.; Schaffert, F. & Feldmann, A. (2014). Panopticon: Reaping the benefits of incremental sdn deployment in enterprise networks. Em *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, pp. 333--346, Berkeley, CA, USA. USENIX Association.

Li, L. E.; Mao, Z. M. & Rexford, J. (2012). Toward software-defined cellular networks. Em *Proceedings of the 2012 European Workshop on Software Defined Networking*, EWSDN '12, pp. 7--12, Washington, DC, USA. IEEE Computer Society.

Mann, V.; Vishnoi, A.; Kannan, K. & Kalyanaraman, S. (2012). Crossroads: Seamless vm mobility across data centers through software defined networking. Em *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pp. 88–96. ISSN 1542-1201.

Martin, J. & Feamster, N. (2012). User-driven dynamic traffic prioritization for home networks. Em *W-MUST 12 Proceedings of the 2012 ACM SIGCOMM workshop on Measurements up the stack*, pp. 19–24.

McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S. & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

Meraki, I. (2009). Meraki whitepaper: Meraki hosted architecture. Accessed: 2015-04-15.

Meraki Networks, I. (2015). Cloud managed wireless. `https://meraki.cisco.com/products/wireless`. Accessed: 2015-05-05.

Meru Networks, I. (2014). Meru whitepaper: Sdn for wi-fi: Openflow-enabling the wireless LAN can bring new levels of agility. Accessed: 2015-04-15.

Moura, H.; Bessa, G. V.; Vieira, M. A. & Macedo, D. F. (2015). Ethanol: Software defined networking for 802.11 wireless networks. Em *IFIP/IEEE International Symposium on Integrated Network Management (IM)*.

Narayanan, R.; Kotha, S.; Lin, G.; Khan, A.; Rizvi, S.; Javed, W.; Khan, H. & Khayam, S. (2012). Macroflows and microflows: Enabling rapid network innovation

through a split sdn data plane. Em *Software Defined Networking (EWSDN), 2012 European Workshop on*, pp. 79–84.

Nunes, B.; Mendonca, M.; Nguyen, X.-N.; Obraczka, K. & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys Tutorials, IEEE*, 16(3):1617–1634. ISSN 1553-877X.

Ozdag, R. (2012). Intel Ethernet Switch FM6000 Series - software defined networking. Relatório técnico, Intel Corporation.

Pan, J.; Paul, S. & Jain, R. (2011). A survey of the research on future internet architectures. *Communications Magazine, IEEE*, 49(7):26–36. ISSN 0163-6804.

Patro, A. & Banerjee, S. (2014). Coap: A software-defined approach for home wlan management through an open api. Em *Proceedings of the 9th ACM Workshop on Mobility in the Evolving Internet Architecture*, MobiArch '14, pp. 31--36.

Pentikousis, K.; Wang, Y. & Hu, W. (2013). Mobileflow: Toward software-defined mobile networks. *Communications Magazine, IEEE*, 51(7):44–53. ISSN 0163-6804.

Perahia, E.; Cordeiro, C.; Park, M. & Yang, L. L. (2010). Ieee 802.11ad: Defining the next generation multi-gbps wi-fi. Em *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, pp. 1--5. IEEE.

Poole, E. S.; Chetty, M.; Grinter, R. E. & Edwards, W. K. (2008). More than meets the eye: transforming the user experience of home network management. Em *DIS '08 Proceedings of the 7th ACM conference on Designing interactive systems*, pp. 455–464.

Riggio, R.; Marina, M.; Schulz-Zander, J.; Kuklinski, S. & Rasheed, T. (2015). Programming abstractions for software-defined wireless networks. *Network and Service Management, IEEE Transactions on*, 12(2):146–162. ISSN 1932-4537.

Rob, P. & Coronel, C. (2002). *Database Systems Design, Implementation and Management*. Course Technology Press, Boston, MA, United States, 5th edição. ISBN 061906269X.

Salvador, P.; Cominardi, L.; Gringoli, F. & Serrano, P. (2013). A first implementation and evaluation of the ieee 802.11 aa group addressed transmission service. *ACM SIGCOMM Computer Communication Review*, 44(1):35--41.

Sezer, S.; Scott-Hayward, S.; Chouhan, P.; Fraser, B.; Lake, D.; Finnegan, J.; Viljoen, N.; Miller, M. & Rao, N. (2013). Are we ready for SDN? implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):36–43.

Shah, S.; Faiz, J.; Farooq, M.; Shafi, A. & Mehdi, S. (2013). An architectural evaluation of sdn controllers. Em *Communications (ICC), 2013 IEEE International Conference on*, pp. 3504–3508. ISSN 1550-3607.

Sherwood, R.; Gibb, G.; Yap, K.-K.; Apenzeller, G.; Casado, M.; McKeown, N. & Parulkar, G. (2009). Flowvisor: A network virtualization layer. Em *Tech. Rep. OPENFLOWTR-2009-1*. OpenFlowSwitch.org.

Shin, M.-K.; Nam, K.-H. & Kim, H.-J. (2012). Software-defined networking (sdn): A reference architecture and open apis. Em *ICT Convergence (ICTC), 2012 International Conference on*, pp. 360–361.

Shirayanagi, H.; Yamada, H. & Kono, K. (2012). Honeyguide: A vm migration-aware network topology for saving energy consumption in data center networks. Em *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pp. 460–467. ISSN 1530-1346.

Sundaresan, S.; Feamster, N.; Teixeira, R.; Grunenberger, Y.; Papagiannaki, D. & Levin, D. (2013). Characterizing home network performance problems.

Suresh, L.; Schulz-Zander, J.; Merz, R.; Feldmann, A. & Vazao, T. (2012). Towards programmable enterprise WLANS with ODIN. Em *HotSDN'12 Proceedings of the first workshop on Hot topics in software defined network*, pp. 115–120.

Taniuchi, K.; Ohba, Y.; Fajardo, V.; Das, S.; Tauil, M.; Cheng, Y.-H.; Dutta, A.; Baker, D.; Yajnik, M. & Famolari, D. (2009). Ieee 802.21: Media independent handover: Features, applicability, and realization. *Comm. Mag.*, 47(1):112--120. ISSN 0163-6804.

Tootoonchian, A. & Ganjali, Y. (2010). Hyperflow: A distributed control plane for openflow. Em *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10.

Verma, L.; Fakharzadeh, M. & Choi, S. (2013). Wifi on steroids: 802.11ac and 802.11ad. *Wireless Communications, IEEE*, 20(6):30–35.

Xia, W.; Wen, Y.; Foh, C.; Niyato, D. & Xie, H. (2015). A survey on software-defined networking. *Communications Surveys Tutorials, IEEE*, 17(1):27–51. ISSN 1553-877X.

Yap, K.-K.; Sherwood, R.; Kobayashi, M.; Huang, T.-Y.; Chan, M.; Handigol, N.; McKeown, N. & Parulkar, G. (2010). Blueprint for introducing innovation into wireless mobile networks. Em *Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '10, pp. 25--32, New York, NY, USA. ACM.

Yiakoumis, Y.; Yap, K.-K.; Katti, S.; Parulkar, G. & McKeown, N. (2011). Slicing home networks. Em *Proceedings of the 2Nd ACM SIGCOMM Workshop on Home Networks*, HomeNets '11, pp. 1--6.

Yin, H.; Xie, H.; Tsou, T.; Lopez, D.; Aranda, P. & Sidi, R. (2012). SDNi: A message exchange protocol for software defined networks (SDNS) across multiple domains. SDNi (Internet-Draft).

Yu, M.; Jose, L. & Miao, R. (2013). Software defined traffic measurement with OpenSketch. Em *USENIX conference on Networked Systems Design and Implementation (NSDI)*, pp. 29--42.

Zheng, P.; Peterson, L. L.; Davie, B. S. & Farrel, A. (2009). *Wireless Networking Complete*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 0123750776, 9780123750778.