

ROGÉRIO LEÃO SANTOS DE OLIVEIRA

**L3-ARPSEC – MÓDULO SEGURO PARA CONTROLE E  
PROTEÇÃO DO PROTOCOLO DE RESOLUÇÃO DE  
ENDEREÇOS EM REDES DEFINIDAS POR SOFTWARE**

ROGÉRIO LEÃO SANTOS DE OLIVEIRA

**L3-ARPSEC – MÓDULO SEGURO PARA CONTROLE E  
PROTEÇÃO DO PROTOCOLO DE RESOLUÇÃO DE  
ENDEREÇOS EM REDES DEFINIDAS POR SOFTWARE**

Qualificação apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Estadual Paulista - UNESP - Câmpus de Ilha Solteira, como parte dos requisitos necessários para obtenção do título de Mestre em Engenharia Elétrica.  
Área de Concentração: Automação.

Prof. Dr. Ailton Akira Shinoda  
Orientador

Profª. Dra. Christiane Marie Schweitzer  
Coorientadora

FICHA CATALOGRÁFICA

Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

Oliveira, Rogério Leão Santos de  
O482L L3-arpsec – módulo seguro para controle e proteção do protocolo de  
resolução de endereços em redes definidas por software / Rogério Leão  
Santos de Oliveira. -- Ilha Solteira: [s.n.], 2015  
96 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de  
Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2015

Orientador: Ailton Akira Shinoda

Co-orientador: Christiane Marie Schweitzer

Inclui bibliografia

1. Redes definidas por software. 2. Protocolo openflow. 3. Segurança de  
redes locais. 4. Envenenamento de cache ARP. 5. Ataque MITM.



**UNIVERSIDADE ESTADUAL PAULISTA**  
CAMPUS DE ILHA SOLTEIRA  
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA

### **CERTIFICADO DE APROVAÇÃO**

**TÍTULO:** L3-ARPSEC - MÓDULO SEGURO PARA CONTROLE E PROTEÇÃO DO PROTOCOLO DE RESOLUÇÃO DE ENDEREÇOS EM REDES DEFINIDAS POR SOFTWARE

**AUTOR:** ROGÉRIO LEÃO SANTOS DE OLIVEIRA

**ORIENTADOR:** Prof. Dr. AILTON AKIRA SHINODA

**CO-ORIENTADORA:** Profa. Dra. CHRISTIANE MARIE SCHWEITZER

Aprovado como parte das exigências para obtenção do Título de Mestre em Engenharia Elétrica ,  
Área: AUTOMAÇÃO, pela Comissão Examinadora:

*Christiane Marie Schweitzer*

Profa. Dra. CHRISTIANE MARIE SCHWEITZER  
Departamento de Matemática / Faculdade de Engenharia de Ilha Solteira

*Ailton Akira Shinoda*

Prof. Dr. ANTONIO MARCOS COSSI  
Departamento de Matemática / Faculdade de Engenharia de Ilha Solteira

*Ed' Wilson Tavares Ferreira*

Prof. Dr. ED' WILSON TAVARES FERREIRA  
Instituto Federal de Educação, Ciência e Tecnologia de Mato Grosso - Campus Cuiabá

Data da realização: 24 de julho de 2015.

A todos que acreditaram e estiveram comigo durante este trabalho.  
DEDICO.

## **AGRADECIMENTOS**

A Deus que nos permite viver com saúde e disposição.

A família pelo apoio e compreensão.

A minha esposa pela constante paciência, incentivo e reconhecimento.

A colega de estudos Professora Msc. Ligia Rodrigues Prete pelo incentivo e compartilhamento de seu conhecimento.

A Professora Dra. Christiane Marie Schweitzer, pela valiosa orientação durante a elaboração deste trabalho.

Ao Professor Dr. Ailton Akira Shinoda, por acreditar em minha capacidade e colaborar quando necessário.

## RESUMO

O protocolo de resolução de endereços (ARP) é usado para mapear endereços IP a endereços MAC em redes locais. Este protocolo possui algumas vulnerabilidades de segurança e uma delas é ataque *Man-in-the-Middle* (MITM), em que o *cache* ARP permite a um *host* interceptar pacotes trocados entre dois outros *hosts*. O conceito de Redes Definidas por Software (SDNs) representam uma abordagem inovadora na área de redes de computadores, uma vez que propõe um novo modelo para o controle de repasse e roteamento dos pacotes de dados que navegam na Internet. Uma das principais características deste novo paradigma é a capacidade de programar funcionalidades nos controladores de rede para gerenciar o tráfego. Este trabalho apresenta o módulo L3-ARPsec, um conjunto de instruções escritas em linguagem de programação Python que propõe uma maneira de controlar a troca de mensagens ARP e também mitigar o ataque MITM em redes locais. O módulo gerencia as requisições e respostas ARP entre todos dispositivos da rede e não permite o envenenamento do *cache* ARP. Depois de apresentados alguns conceitos do paradigma SDN, a estrutura do protocolo ARP e como o ataque MITM ocorre, o módulo L3-ARPsec é explicado em detalhes e os resultados de diversos testes executados são mostrados.

**Palavras-chave:** Redes definidas por software. *OpenFlow*. Envenenamento de *cache* ARP. MITM.

## ABSTRACT

The Address Resolution Protocol (ARP) is used to map IP addresses to MAC addresses in local area networks. This protocol has some security vulnerabilities and one of them is the Man-in-the-Middle (MITM) attack, a way to poisoning the ARP cache that allows a host to intercept packets switched between two other hosts. Software-Defined Networks (SDNs) represent an innovative approach in the area of computer networks, since they propose a new model to control forwarding and routing data packets that navigate the World Wide Web. One of the main features of this new paradigm is the ability to program functionalities in network controllers to manage the traffic. This study presents the module L3-ARPSec, a set of instructions written in the Python programming language that proposes a way to control the switching of ARP messages and also mitigates the MITM attack in local area networks. The module manages the ARP request, reply messages between all network devices and does not permit the ARP cache poisoning. After presenting some concepts of the SDN paradigm, the ARP protocol structure and how MITM attacks occurs, the L3-ARPSec module is explained in detail and the results of several tests performed are displayed.

**Keywords:** Software-defined network. *OpenFlow*. ARP cache poisoning. MITM.



## LISTA DE FIGURAS

Figura 1 – Estrutura geral da Internet e seus componentes. ....	19
Figura 2 – Analogia em protocolos humanos e de redes. ....	20
Figura 3 – Encapsulamento e comunicação entre as camadas de rede. ....	22
Figura 4 – Roteamento e repasse na camada de rede. ....	26
Figura 5 – Modelo de referência OSI, TCP/IP e Internet. ....	28
Figura 6 – Estrutura geral de uma SDN e seus componentes. ....	30
Figura 7 – Exemplo de uma entrada na tabela de fluxos <i>OpenFlow</i> . ....	32
Figura 8 – Exemplos de uso de um <i>Switch OpenFlow</i> . ....	33
Figura 9 – Exemplos de entradas e ações na tabela de fluxos. ....	35
Figura 10 – Exemplo de protótipo SDN no Mininet. ....	38
Figura 11 – <i>Script</i> console.py monitorando os <i>hosts</i> da rede. ....	40
Figura 12 – Ferramenta MiniEdit utilizada para editar a topologia da rede. ....	40
Figura 13 – Comando para iniciar console gráfico. ....	41
Figura 14 – Executar a ferramenta <i>wireshark</i> no console gráfico. ....	41
Figura 15 – Ferramenta <i>wireshark</i> em execução. ....	42
Figura 16 – Oracle VM com o Mininet e o POX instalados. ....	45
Figura 17 – Topologia do cenário de simulação. ....	46
Figura 18 – Comandos e resultados no cenário. ....	47
Figura 19 – Regras aplicadas nas tabelas de fluxo dos <i>switches</i> . ....	48
Figura 20 – Topologia com 15 nós. ....	49
Figura 21 – Gráfico de tempo gasto para criação e destruição das redes. ....	50
Figura 22 – Gráfico representando a memória ocupada para as redes. ....	51
Figura 23 – Topologia de uma rede local. ....	57
Figura 24 – <i>Host</i> atacante em uma rede local. ....	58
Figura 25 – Cenário para realização de testes. ....	59
Figura 26 - Fluxograma do módulo L3-ARPSec. ....	64
Figura 27 – Dados coletados durante tentativa de ataque. ....	69
Figura 28 – Dados coletados durante tentativa de ataque. ....	72
Figura 29 – Estrutura tradicional para execução de teste. ....	74
Figura 30 – Tela de configuração do IPERF. ....	75
Figura 31 – Velocidade de transmissão na estrutura tradicional. ....	76
Figura 32 – Estrutura SDN para execução de teste. ....	77

Figura 33 – Velocidade de transmissão na estrutura SDN. ....	78
Figura 34 – Estrutura tradicional e SDN. ....	79
Figura 35 – Fluxo de pacotes registrado no teste. ....	79

## LISTA DE TABELAS

Tabela 1 – Resultados obtidos nos testes de escalabilidade. ....	49
Tabela 2 – Resultados obtidos por outro trabalho similar. ....	51
Tabela 3 – Exemplo de uma tabela ARP. ....	55
Tabela 4 – Estrutura das tabelas virtuais. ....	63
Tabela 5 – Tabela contendo a quantidade de respostas ARP por endereço IP. ....	66
Tabela 6 – Entradas fraudulentas detectadas na tabela virtual. ....	71
Tabela 7 – Tabela de punição adaptativa. ....	73

## LISTA DE SIGLAS

API	Application Program Interface
ARP	Address Resolution Protocol
DDos	Distributed Deny of Service
DNS	Domain Name System
DoS	Deny of Service
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol (Protocolo da Internet)
ISO	International Standards Organization
LANs	Local Area Networks
MAC	Medium Access Control
MITM	Man-in-the-Middle
OSI	Open Systems Interconnection
PC	Personal Computer
PDA	Personal Digital Assistants
PPP	Point-to-point Protocol
RAM	Random Access Memory
SBRC	Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos
SDNs	Software Defined networks
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
VLANs	Virtual Local Area Networks
VM	Virtual Machine

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>14</b>
1.1	OBJETIVO GERAL .....	16
1.2	CONTRIBUIÇÃO DO TRABALHO .....	16
1.3	ORGANIZAÇÃO DO TEXTO.....	16
<b>2</b>	<b>REFERENCIAL TEÓRICO .....</b>	<b>18</b>
2.1	REDES DE COMPUTADORES E A INTERNET .....	18
2.1.1	<b>Protocolos de rede .....</b>	<b>19</b>
2.1.2	<b>Camadas de protocolos e seus serviços.....</b>	<b>20</b>
2.1.2.1	<i>Camada de aplicação.....</i>	<i>22</i>
2.1.2.2	<i>Camada de transporte .....</i>	<i>23</i>
2.1.2.3	<i>Camada de rede .....</i>	<i>23</i>
2.1.2.4	<i>Camada de enlace.....</i>	<i>26</i>
2.1.2.5	<i>Camada física .....</i>	<i>27</i>
2.1.3	<b>Os modelos de referência OSI, TCP/IP e internet.....</b>	<b>27</b>
2.2	REDES DEFINIDAS POR SOFTWARE (Software Defined Networks - SDNs) .....	29
2.2.1	<b>Os elementos das SDNs .....</b>	<b>30</b>
2.2.2	<b>Estrutura e comunicação entre os elementos.....</b>	<b>32</b>
2.3	FERRAMENTAS PARA SIMULAÇÃO E TESTES SDNs.....	36
2.3.1	<b>O emulador MININET .....</b>	<b>36</b>
2.3.1.1	<i>Elementos de SDN no Mininet .....</i>	<i>37</i>
2.3.1.2	<i>Interatividade com a rede do protótipo e ferramentas de controle.....</i>	<i>38</i>
2.3.1.3	<i>O emulador e os controladores SDNs.....</i>	<i>42</i>
2.3.1.4	<i>Simulando SDN com Mininet e POX.....</i>	<i>44</i>
2.3.1.5	<i>Especificação do ambiente de emulação .....</i>	<i>44</i>
2.3.2	<b>Testes de funcionamento – 2 hosts, 2 switches e 1 controlador.....</b>	<b>45</b>
2.3.2.1	<i>Testes de escalabilidade.....</i>	<i>48</i>
2.3.2.2	<i>Resultados e discussão .....</i>	<i>52</i>
2.3.2.3	<i>Limitações.....</i>	<i>52</i>
2.3.2.4	<i>Prototipação.....</i>	<i>53</i>
2.3.2.5	<i>Aplicabilidade .....</i>	<i>53</i>
2.3.2.6	<i>Compartilhamento.....</i>	<i>53</i>
2.3.3	<b>Conclusão .....</b>	<b>54</b>
<b>3</b>	<b>O PROBLEMA DE SEGURANÇA DO PROTOCOLO DE RESOLUÇÃO DE ENDEREÇOS (ARP) .....</b>	<b>55</b>
3.1	O PROTOCOLO DE RESOLUÇÃO DE ENDEREÇOS (ARP) .....	55
3.2	ATAQUES DE ENVENENAMENTO DE CACHE ARP E O ATAQUE MITM.....	57
3.3	REDES DEFINIDAS POR SOFTWARE E O ATAQUE MITM .....	58

3.3.1	Cenário de testes.....	59
3.3.2	Conclusão .....	60
4	<b>O MÓDULO L3-ARPSEC .....</b>	<b>62</b>
4.1	TABELAS VIRTUAIS DE MAPEAMENTO DE ENDEREÇOS NO CONTROLADOR.....	62
4.2	ENVIO DE TODO PACOTE PARA A FUNÇÃO DE AUTO-APRENDIZADO .....	63
4.3	VERIFICAÇÃO SE O PACOTE É DO TIPO IP OU ARP.....	65
4.4	TIMERS DE VERIFICAÇÃO .....	67
4.5	EXECUÇÃO DE TESTES COM O MÓDULO L3-ARPSEC .....	67
4.5.1	Testando a função de detecção de inundação ARPReplayFlood .....	68
4.5.2	Resultados e discussão do primeiro teste .....	69
4.5.3	Testando os Timers para a detecção dos ataques MITM.....	70
4.5.4	Resultados e discussão do segundo teste .....	71
4.5.5	Tempo de bloqueio adaptativo para ataque recorrente .....	72
4.5.6	Avaliando o desempenho do módulo .....	73
4.5.6.1	<i>Desempenho Utilizando uma Rede Tradicional.....</i>	<i>74</i>
4.5.6.2	<i>Desempenho utilizando uma rede open flow e o módulo L3-ARPSec.....</i>	<i>76</i>
4.5.6.3	<i>Desempenho e funcionalidade com vários usuários.....</i>	<i>78</i>
5	<b>RESULTADOS E DISCUSSÕES .....</b>	<b>81</b>
6	<b>CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS .....</b>	<b>82</b>
	<b>REFERÊNCIAS .....</b>	<b>83</b>
	<b>APÊNDICES.....</b>	<b>85</b>

## 1 INTRODUÇÃO

É inegável o imenso sucesso das redes de computadores atualmente. Todas as atividades da sociedade, direta ou indiretamente utilizam recursos ou serviços oferecidos pelas redes de comunicação.

A Internet é uma rede mundial de computadores que interconecta dispositivos computacionais ao redor do mundo. Até há pouco tempo, esses dispositivos eram basicamente computadores de mesa, estações de trabalho, e servidores de dados e serviços que armazenam e transmitem informações, como páginas da Web e mensagens de e-mail. No entanto, cada vez mais sistemas finais modernos da Internet, como TVs, *laptops*, consoles para jogos, telefones celulares, *webcams*, automóveis, dispositivos de sensoriamento ambiental, quadros de imagens, e sistemas internos elétricos e de segurança, estão sendo conectados à rede. Realmente, o termo *rede de computadores* está começando a soar um tanto desatualizado, dados os muitos equipamentos não tradicionais que estão sendo ligados à Internet. No jargão da Internet, todos esses equipamentos são denominados hospedeiros ou sistemas finais (KUROSE; ROSS, 2010).

Esta rede mundial de computadores, conhecida como Internet, ao mesmo tempo em que proporciona comunicação às diversas regiões do mundo promovendo até certa dependência para os usuários, também enfrenta problemas devido ao nível de amadurecimento em sua arquitetura, tornou-se pouco flexível. Novas implementações e novas tecnologias sempre dependem de alteração de hardware (roteadores e *switches*), o que é muito custoso e às vezes inviáveis se frequentes.

A comunidade de pesquisa em redes de computadores vem procurando soluções que possibilitem a utilização de redes com maiores recursos de programação e menor necessidade de troca de elementos de hardware, de forma que novas tecnologias criadas para solucionar novos problemas possam ser inseridas na rede de forma gradual e sem custos tão significativos.

Uma das iniciativas mais bem-sucedidas nesse sentido é a definição da interface e do protocolo *OpenFlow* (MCKEOWN et al., 2008). Com ele, os elementos que encaminham os pacotes oferecem uma interface de programação simplificada que lhes permite estender o acesso e controle da tabela de consulta utilizada pelo hardware para determinar o próximo passo de repasse de cada pacote recebido. Sendo assim, o hardware ainda continua fazendo a tarefa de encaminhamento e repasse garantindo a eficiência atual, mas deixando a decisão de como cada pacote será processado, para uma camada superior, onde diferentes

funcionalidades podem ser configuradas. A essa estrutura controlada por aplicações, ou este novo paradigma, dá-se o nome de redes definidas por software, ou *Software Defined Networks* (SDNs).

Esta nova estrutura controlada por aplicações gerencia a transferência de pacotes na rede, mas não interfere nos atuais protocolos da camada de rede, como *Address Resolution Protocol* (ARP), *Internet Protocol* (IP) e *Transmission Control Protocol* (TCP). Assim, problemas de segurança já conhecidos que afetam estes protocolos na atual estrutura de rede, também existem na nova estrutura SDN. Um deles é o ataque chamado *Man-in-the-Middle* (MITM), em que um atacante usando envenenamento de *cache* ARP, fica no meio da comunicação entre dois *hosts* e depois de estabelecido e configurado o ataque, ele consegue visualizar todas as mensagens trocadas entre as vítimas.

Assim, o principal objetivo deste trabalho é usufruir da possibilidade de programação de funcionalidades concebida por esta nova estrutura de redes definidas por software e criar um mecanismo de segurança capaz de defender as redes locais do ataque MITM.

Outros trabalhos propuseram soluções para o problema dos ataques ao protocolo ARP, dentre eles é interessante ressaltar alguns trabalhos. Bruschi et al. (2003) e Lootah et al. (2005) usam mecanismos de criptografia, exigindo a qualquer *host*, que queira entrar na rede local, a geração de um par de chaves digitais assimétricas, uma pública e outra privada e desta forma todas as mensagens de requisição e resposta do protocolo ARP seriam autenticadas digitalmente. Esta solução, no entanto, possui significativa redução de desempenho, exigindo esforço inicial por parte dos administradores de rede para emitir e validar os certificados digitais para cada um dos novos *hosts* e não sendo totalmente compatível com o protocolo padrão ARP.

Outra abordagem foi proposta por Tripunitara e Dutta (1999), a qual não era praticável, uma vez que requeria alterações em todos os *hosts* da rede. A solução proposta por Gouda e Huang (2003), foi a mais ambiciosa, mas também exigia alteração em todos os *hosts* da rede, complexas configurações e bastante limitada a redes estáticas.

Os trabalhos encontrados acerca do assunto propuseram soluções baseadas em técnicas e configurações de redes, não fazendo uso das SDNs. Desta forma, este trabalho representa uma abordagem inovadora frente a um problema antigo das redes de computadores locais.



## 1.1 OBJETIVO GERAL

Este trabalho propõe e apresenta um módulo de programação para controladores de rede SDN, cujo objetivo é gerenciar as mensagens ARP e mitigar o ataque MITM em redes locais.

É importante ressaltar que a abrangência desta proposta é limitada a redes locais (LANs) e também se pressupõe que a conexão entre controlador e comutador é segura e confiável.

## 1.2 CONTRIBUIÇÃO DO TRABALHO

Os resultados desta pesquisa poderão ser utilizados para inovação e desenvolvimento da ciência computacional e da comunicação entre dispositivos eletrônicos. O meio industrial e comercial da área de redes de comunicação, também poderá ser beneficiado com este trabalho, visto que novos equipamentos ou softwares poderão surgir e serem utilizados em todo o mundo.

Com a divulgação deste trabalho por meio de participações em congressos e publicações em periódicos, esta instituição também se fortalecerá como difusora de conhecimento, avanço tecnológico e pesquisa científica.

## 1.3 ORGANIZAÇÃO DO TEXTO

O presente trabalho está organizado da seguinte forma:

No capítulo 2 é apresentado um panorama geral da estrutura tradicional das redes de computadores e da Internet, detalhando os principais elementos que a compõem e maneira como eles se interagem para permitir o repasse e roteamento dos dados. É também apresentada neste capítulo uma abordagem do paradigma de SDN, descrevendo a motivação de seu surgimento, os elementos de redes que fazem parte desta nova estrutura e o funcionamento detalhado destes componentes comparando-os à estrutura tradicional. Ainda nesta seção é demonstrada a ferramenta de simulação de SDNs chamada *Mininet* e alguns controladores de SDNs. Posteriormente são executados alguns testes para avaliar essa ferramenta e os resultados são discutidos e apresentados.

O conteúdo do capítulo 3 retrata o problema relacionado ao ataque de segurança utilizando o protocolo ARP em redes de computadores locais, alvo deste trabalho. É explicado

também o funcionamento do protocolo ARP em operações normais e como o ataque MITM ocorre em detalhes utilizando o protocolo ARP na estrutura atual de redes de computadores. No final desta seção um cenário SDN é criado para testar o ataque na prática e servir de base para os testes das seções seguintes.

É proposto no capítulo 4 o módulo L3-ARPsec, juntamente com suas funcionalidades, sua estrutura e como se pretende controlar a troca de mensagens ARP e mitigar o ataque MITM. Ainda neste capítulo são realizados alguns testes com o módulo e seus resultados são apresentados e discutidos.

No capítulo 5 são apresentados os resultados e as discussões, e finalmente no capítulo 6 as conclusões e sugestões de trabalhos futuros são mostradas.

## 2 REFERENCIAL TEÓRICO

Este capítulo tem por objetivo apresentar os principais conceitos abordados neste trabalho, desde os fundamentos básicos até a estrutura das redes de comunicação atual. Ainda neste capítulo, o paradigma de redes definidas por software, os novos elementos e suas interações serão também apresentados.

### 2.1 REDES DE COMPUTADORES E A INTERNET

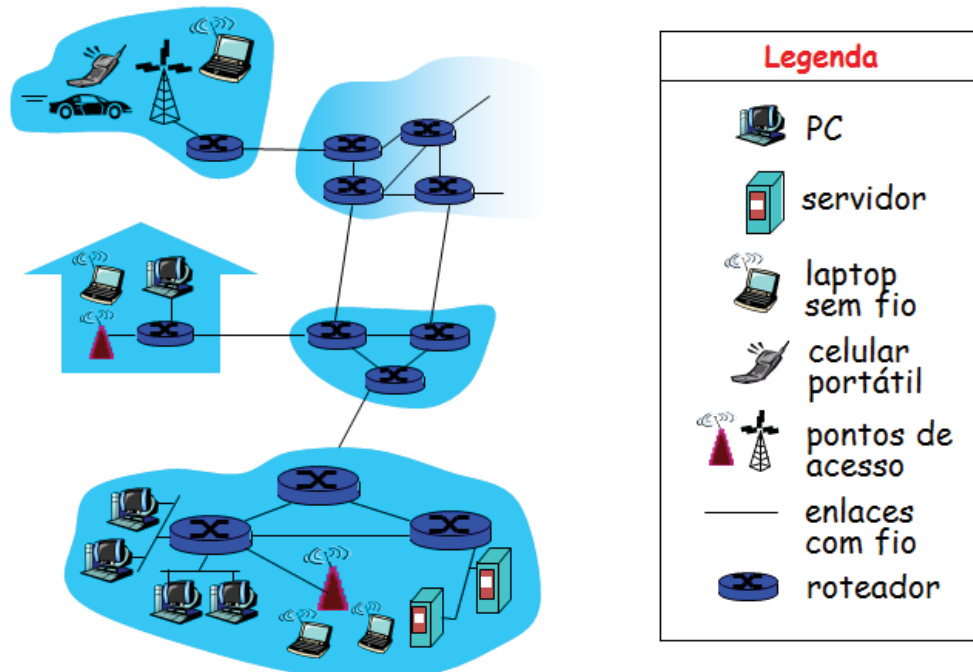
A Internet de hoje é um importante sistema de engenharia criado pela humanidade, com centenas de dispositivos conectados, enlaces de comunicação e comutadores; centenas de milhares de usuários que se conectam esporadicamente por meio de telefones celulares e assistentes digitais pessoais (PDAs, *Personal Digital Assistants*); e dispositivos como sensores *webcams*, console para jogos, quadros de imagens, e até mesmo eletrodomésticos conectados à Internet (KUROSE; ROSS, 2010).

Em uma visão mais detalhada, a estrutura da Internet pode ser dividida em duas partes: componentes de hardware e software. Os componentes de hardware são de diversos tipos e desempenham funções variadas. Na periferia da Internet encontram-se os dispositivos finais, *tablets*, microcomputadores, *laptops* e outros aparelhos que são utilizados pelos usuários para consumir serviços como jogos, notícias, vídeos e outros, de grandes servidores que geralmente se encontram do outro lado desta periferia. No núcleo da rede estão os componentes de roteamento e repasse de dados, são roteadores e *switches* que interligados por diferentes meios físicos retransmitem os pacotes de dados e propiciam a comunicação entre os dispositivos finais (TANENBAUM; WETHERALL, 2011). Estes componentes de hardware e sua estrutura de interligação podem ser observados na Figura 1.

Numa visão de serviço, cada vez mais, os avanços na tecnologia dos componentes da Internet estão sendo guiados pelas necessidades de novas aplicações. Portanto é importante ter em mente que a Internet é uma infraestrutura na qual, novas aplicações estão constantemente sendo inventadas e disponibilizadas. Assim, os componentes de software desempenham neste contexto um papel imprescindível, eles são responsáveis por toda a lógica de transferência das informações entre os dispositivos de hardware. Eles definem a semântica, o formato, a ordem, a sintaxe das mensagens de serviços trocadas entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão ou no recebimento de uma mensagem ou outro evento. Os protocolos de rede, como também são conhecidos, governam também toda a

maneira como os pacotes de dados são retransmitidos pelos roteadores do núcleo da rede até conseguirem alcançar seu destino (KUROSE; ROSS, 2010).

**Figura 1** – Estrutura geral da Internet e seus componentes.



Fonte: Elaboração do autor.

### 2.1.1 Protocolos de rede

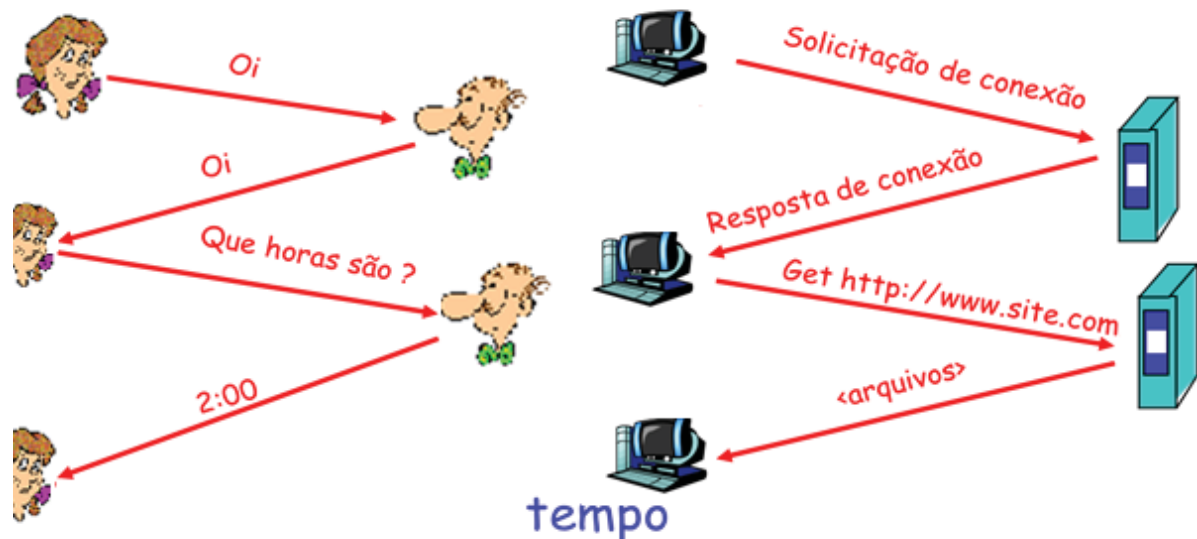
Depois de compreendidos os componentes de hardware e software que formam as redes de computadores, é necessário detalhar um item fundamental para o funcionamento da Internet: os *protocolos de rede*.

Basicamente, um protocolo é um acordo entre as partes que se comunicam, estabelecendo como se dará a comunicação. Como analogia, quando uma mulher é apresentada a um homem, ela pode estender a mão para ele que, por sua vez, pode apertá-la ou beijá-la, dependendo, por exemplo, do fato de ela ser uma advogada norte-americana que esteja participando de uma reunião de negócios ou uma princesa europeia presente em um baile de gala. A violação do protocolo dificultará a comunicação, se não a tornar completamente impossível (TANENBAUM; WETHERALL, 2011).

Como ilustrado na Figura 2, um protocolo de rede é então semelhante a um protocolo humano; a única diferença é que as entidades que trocam mensagens e realizam ações são componentes de hardware ou software de algum equipamento (por exemplo, computador,

PDA, telefones celulares, roteador ou outro equipamento habilitado para rede). Todas as atividades na Internet que envolvem duas ou mais entidades remotas comunicantes são governadas por um protocolo. Por exemplo, protocolos implementados em hardware nas placas de interface de rede de dois computadores conectados fisicamente controlam o fluxo de *bits* no ‘cabo’ entre as duas placas de interface de rede; protocolos de congestionamento em sistemas finais controlam a taxa com que os pacotes são transmitidos entre a origem e o destino; protocolos em roteadores determinam o caminho de um pacote da origem ao destino (KUROSE; ROSS, 2010).

**Figura 2** – Analogia em protocolos humanos e de redes.



Fonte: Kurose e Ross (2010).

### 2.1.2 Camadas de protocolos e seus serviços

Para reduzir a complexidade de seu projeto, a maioria das redes é organizada como uma pilha de **camadas** (ou níveis), colocadas umas sobre as outras. O número, o nome, o conteúdo e a função de cada camada diferem de uma rede para outra. No entanto, em todas as redes o objetivo de cada camada é oferecer determinados serviços às camadas superiores, isolando essas camadas dos detalhes de implementação real desses recursos. Em certo sentido, cada camada é uma espécie de máquina virtual, oferecendo determinados serviços à camada situada acima dela (TANENBAUM; WETHERALL, 2011).

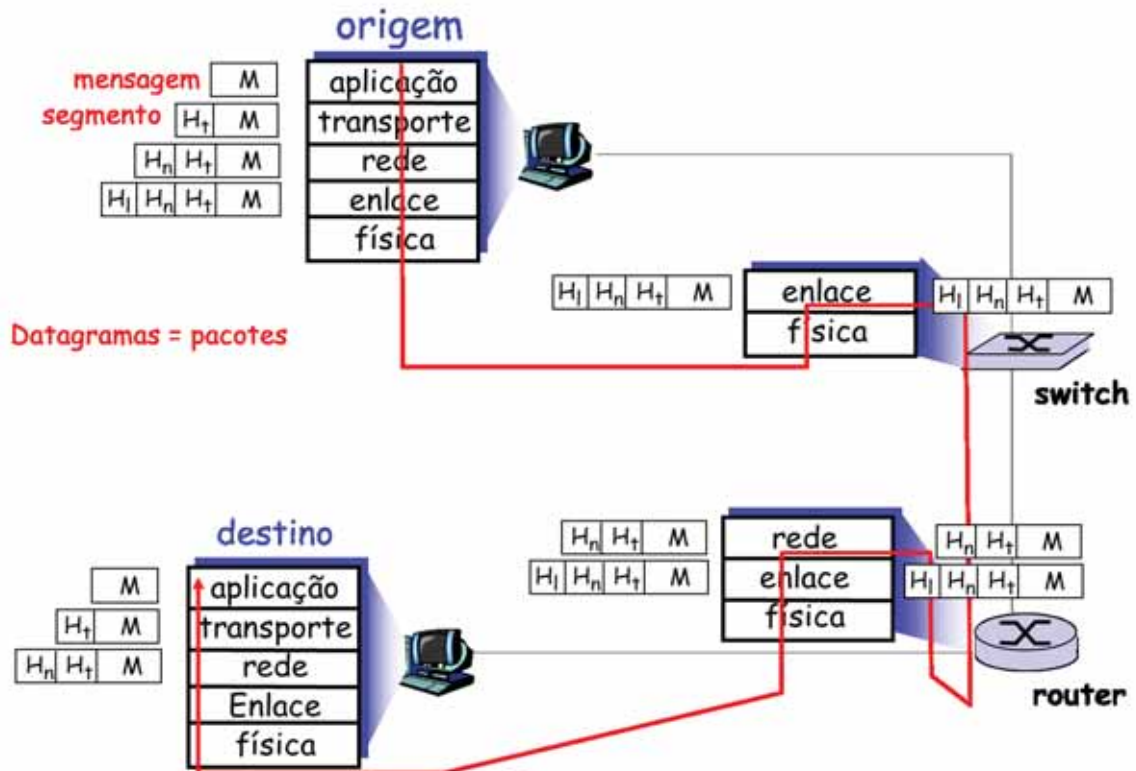
Este conceito é muito familiar na ciência da computação, na qual é conhecido por nomes diferentes, como ocultação de informações, tipos de dados abstratos, encapsulamento de dados e programação orientada a objetos. A ideia fundamental é que um determinado item de software (ou hardware) forneça um serviço a seus usuários, mas mantenha ocultos os detalhes de seu estado interno e de seus algoritmos. No funcionamento das redes de computadores, cada camada presta um serviço específico para outra camada.

Em cada camada de rede existe um protocolo que é o responsável por executar sua tarefa, padronizar e organizar o repasse dos dados para a camada seguinte. Este processo de repasse entre as camadas, chamado encapsulamento, é feito sequencialmente do início da comunicação, quando uma aplicação rodando em um determinado dispositivo qualquer queira se comunicar com outra, até o momento em que esta mensagem chega à aplicação no dispositivo de destino.

As camadas de redes de computadores definidas em um modelo de referência chamado Protocolo de Controle de Transmissão/Protocolo de Internet (TCP/IP, *Transmission Control Protocol/Internet Protocol*), são: aplicação, transporte, rede, enlace e física. Cada uma delas realiza um serviço diferente e utiliza seus protocolos para padronizar a comunicação. Ao receberem da camada anterior um pacote, elas realizam suas tarefas, adicionam seu cabeçalho e repassam novamente o pacote encapsulado para próxima camada.

Na Figura 3 é ilustrado o processo de encapsulamento e comunicação entre as camadas de rede deste modelo.

**Figura 3** – Encapsulamento e comunicação entre as camadas de rede.



Fonte: Kurose e Ross (2010).

Nas próximas subseções serão explicadas cada camada de rede e suas respectivas finalidades.

### 2.1.2.1 Camada de aplicação

A camada de aplicação é onde residem aplicações de rede e seus protocolos. Ela inclui muitos protocolos, tais como o protocolo (*hypertext transfer protocol* – HTTP), que provê requisição e transferência de documentos pela web, o (*simple mail transfer protocol* – SMTP), que provê transferência de mensagens de correio eletrônico e o (*file transfer protocol* – FTP), que prove a transferência de arquivos entre dois sistemas finais. Algumas funções de rede, como a tradução de nomes fáceis de entender dados a sistemas finais da Internet (por exemplo, [www.unesp.br](http://www.unesp.br)) para um endereço de rede de 32 bits, também são executadas com a ajuda de um protocolo de camada de aplicação, no caso, o sistema de nomes de domínio (*domain name system* – DNS).

Uma aplicação quando deseja se comunicar com outra pela rede, envia suas mensagens (como são chamados os pacotes de dados desta camada) para a camada inferior a

ela chamada de camada de transporte. No destino, a aplicação que receber esta mensagem deverá utilizar o mesmo protocolo adotado no emissor para conseguir interpretar a comunicação e conseguir realizar alguma ação.

### **2.1.2.2 Camada de transporte**

A camada de transporte da Internet transporta mensagens da camada de aplicação entre as entidades comunicantes. Há dois protocolos de transporte na Internet: TCP (*Transmission Control Protocol*) e UDP (*User Datagram Protocol*), e qualquer um deles pode levar mensagens de camada de aplicação. O TCP provê serviços orientados para conexão para suas aplicações. Dentre estes serviços há a entrega garantida de mensagens da camada de aplicação ao destino e controle de fluxo (isto é, compatibilização das velocidades do remetente e do receptor). O TCP também fragmenta mensagens longas em segmentos mais curtos e provê mecanismo de controle de congestionamento, de modo que uma origem regula sua velocidade de transmissão quando a rede está congestionada. O protocolo UDP provê serviço não orientado a conexão em suas aplicações. Este é um serviço econômico que fornece segurança, sem controle de fluxo e de congestionamento (KUROSE; ROSS, 2010).

Os pacotes de dados desta camada são chamados de *segmentos*. Esta camada recebe a mensagem da camada de aplicação, executa seu serviço com ou sem controle de conexão, dependendo do protocolo adotado para o transporte dos dados, acrescenta seu cabeçalho e repassa um ou mais segmentos para a próxima camada, a de rede.

### **2.1.2.3 Camada de rede**

O serviço desempenhado por esta camada, de forma semelhante a camada anterior, também pode ser orientado a conexão ou não. As redes de computadores que oferecem o serviço orientado a conexão são chamadas redes de circuitos virtuais e as redes que oferecem o serviço não orientado a conexão são chamadas de redes de datagramas. Um circuito virtual consiste em um caminho entre origem e destino (mesmo que através de vários roteadores e enlaces), números de circuitos virtuais (um número para cada enlace ao longo do caminho) e registros na tabela de repasse dos roteadores ao longo do caminho. Nestas redes, os pacotes carregam em seus respectivos cabeçalhos o número do circuito virtual a qual pertencem. Estes números podem ser alterados nos roteadores de acordo com a tabela de roteamento de circuitos virtuais.



De uma forma bem simples e didática, é possível fazer uma analogia das redes de circuitos virtuais com o sistema de telefonia tradicional. Quando uma chamada é estabelecida entre dois pontos, recursos de transferência de voz são alocados nos diversos enlaces intermediários, deixando a ligação exclusiva. Uma desvantagem deste sistema dedicado, é o possível desperdício de recurso, uma vez que, se outra pessoa em um dos dois pontos anterior quiser falar com qualquer outro ponto, terá que esperar a ligação terminar. Nas redes de circuito isto também ocorre, os roteadores de núcleo reservam recursos e quando não há dados para transferir continuam com a reserva estabelecida e ociosos.

Por sua vez, as redes de *datagramas* não reservam recursos dedicados, portanto, compartilham melhor o meio físico de transmissão permitindo que vários comunicantes troquem pacotes ao mesmo tempo, mas não possuem um serviço de entrega de dados garantida. A proposta deste trabalho está baseada neste tipo de rede de *datagramas* e, portanto, mais detalhes sobre ela serão dados a seguir.

A camada de rede é a responsável pela movimentação, de uma máquina para outra, dos pacotes, nesta camada conhecidos como *datagramas*. O protocolo de camada de transporte da Internet (TCP ou UDP) em uma máquina de origem passa um segmento de camada de transporte e um endereço de destino à camada de rede, exatamente como se passa uma carta com endereço de destino ao serviço de correios.

Nesta camada são realizados dois serviços de suma importância para o sucesso da transmissão dos dados na Internet: o endereçamento e o roteamento de pacotes.

O protocolo denominado IP é o responsável pelo endereçamento, ele define um endereço de 32 *bits* a cada interface de rede dos dispositivos e cuida para que estes sejam exclusivos. Estes valores de endereços são utilizados como base para o repasse e roteamento de cada *datagrama* durante todo o trajeto desde a saída do dispositivo emissor até a chegada ao dispositivo receptor.

Outro componente muito importante desta camada é o protocolo de roteamento que determina as rotas que os *datagramas* seguem entre origens e destinos. A camada de transporte entrega o segmento para esta camada de rede que de acordo com as regras definidas pelo protocolo de roteamento, adiciona seu cabeçalho e repassa o *datagrama* para a próxima camada, chamada de *camada de enlace*.

Na Internet, cada pacote que atravessa a rede contém o seu endereço de destino em seu cabeçalho. Como os endereços postais, esse endereço possui uma estrutura hierárquica. Quando um pacote chega a um roteador na rede, o roteador examina uma parte do endereço de destino do pacote e conduz o pacote a um roteador adjacente. Especificamente falando,

cada roteador possui uma base de encaminhamento que mapeia o endereço de destino (ou partes desse endereço) para endereços de saída. Quando um pacote chega ao roteador, este o examina e busca sua base utilizando o endereço de destino para encontrar o enlace de saída apropriado (KUROSE; ROSS, 2010).

Esta base de encaminhamento do roteador é também chamada de tabela de rotas e seu conteúdo pode ser de duas formas básicas. No roteamento estático, os valores são definidos pelo administrador de redes e nunca se alteram, assim um pacote recebido em uma determinada interface do roteador sempre será repassado para outra determinada interface. Diferentemente disto, no roteamento dinâmico os valores da tabela de rotas podem ser alterados por diversas situações dependendo do protocolo de roteamento empregado (KUROSE; ROSS, 2010).

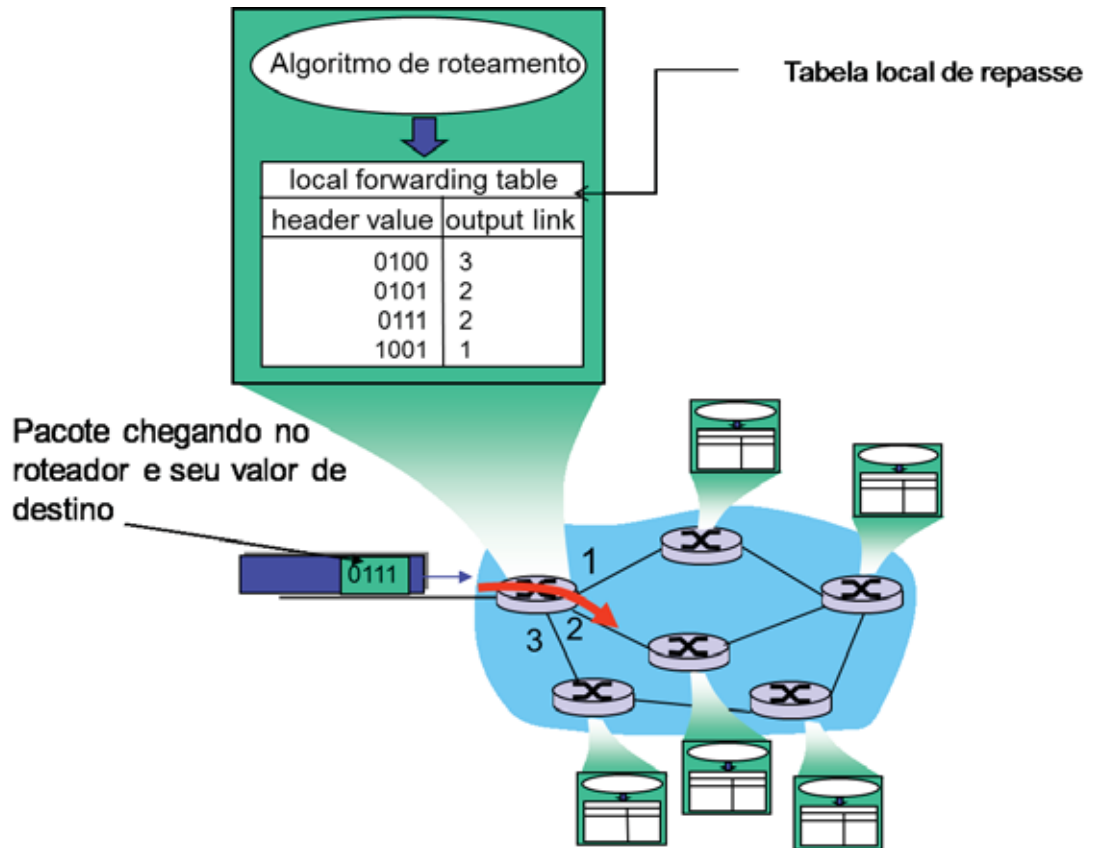
Os algoritmos de roteamento podem ser agrupados em duas classes principais: não adaptativos e adaptativos. Os algoritmos não adaptativos não baseiam suas decisões de roteamento em medidas ou estimativas do tráfego. Em vez disso, a escolha da rota a ser utilizada para ir de I até J (para todo I e todo J) é previamente calculada *off-line*, sendo transferida para os roteadores quando a rede é iniciada. Às vezes esse procedimento é chamado de roteamento estático. Por não responder bem a falhas, o roteamento estático é mais útil para situações em que a escolha de rotas é óbvia (TANENBAUM; WETHERALL, 2011).

Em contraste, os algoritmos adaptativos alteram as decisões de roteamento para refletir mudanças na topologia e, normalmente, também no tráfego. Esses algoritmos de roteamento dinâmico diferem em termos do lugar em que obtêm suas informações (por exemplo, no local, de roteadores adjacentes ou de todos os roteadores), do momento em que alteram as rotas (por exemplo, quando a topologia muda ou a cada intervalo  $\Delta T$  segundos, quando a carga se altera) e da métrica utilizada na otimização (por exemplo, distância, número de saltos ou tempo estimado de tráfego) (TANENBAUM; WETHERALL, 2011).

Os roteadores ficam todo tempo trocando informações entre si do estado de seus enlaces para saber se determinada rota está congestionada, por exemplo, ou para saber a distância entre seus enlaces. Assim, baseado na política de roteamento imposta pelo protocolo de roteamento ativo, os roteadores alteram suas bases de encaminhamento para permitir que os pacotes de dados sejam roteados de forma mais eficiente, pelo menor caminho ou pelo caminho menos congestionado (KUROSE; ROSS, 2010).

Um exemplo de roteamento e repasse é ilustrado na Figura 4.

Figura 4 – Roteamento e repasse na camada de rede.



Fonte: Kurose e Ross (2010).

#### 2.1.2.4 Camada de enlace

A camada de rede da Internet roteia um *datagrama* por meio de uma série de roteadores entre a origem e o destino. Para levar um pacote de um nó (sistema final ou comutador de pacotes) ao nó seguinte na rota, a camada de rede depende dos serviços da camada de enlace. Em especial, em cada nó a camada de rede passa o *datagrama* para a camada de enlace, que o entrega, ao longo da rota, ao nó seguinte, no qual o *datagrama* é repassado da camada de enlace para a de rede (KUROSE; ROSS, 2010).

Nesta camada, a prestação dos serviços depende do protocolo empregado. São vários os tipos de protocolos que podem ser utilizados, PPP (*point-to-point* – protocolo ponto a ponto), Ethernet (protocolo multiponto para redes locais) e outros, que podem prover ou não garantias de entrega e detecção de erros dos quadros transmitidos de um nó a outro. A ligação física entre um nó e outro é chamada de enlace (*link*), independentemente do meio físico que é empregado para transportar os *bits*. Cada enlace destes pode implementar um protocolo

diferente dos outros durante o trajeto todo de um *datagrama* que pode passar por vários nós até alcançar o seu destino final.

Os quadros, como são chamados os pacotes de dados desta camada, assumem formatos diferentes de acordo com o protocolo de acesso ao meio definido para o enlace.

#### **2.1.2.5 Camada física**

Enquanto a tarefa da camada de enlace é movimentar quadros inteiros de um elemento da rede até um elemento adjacente, a da camada física é movimentar os *bits* individuais que estão dentro do quadro de um nó para o seguinte. Os protocolos nessa camada novamente dependem do enlace e, além disso, dependem do próprio meio de transmissão do enlace (por exemplo, fios de cobre trançado ou fibra ótica *monomodal*). Por exemplo, a Ethernet tem muitos protocolos de camada física: um para par de fios de cobre trançado, outro para cabo coaxial, outro para fibra e assim por diante. Em cada caso, o *bit* é movimentado pelo enlace de um modo diferente.

Todos os dados que são transportados de uma origem até um destino, quando chegam a esta camada são codificados em um formato binário (zeros ou uns), e podem ser representados de maneiras diferentes de acordo com o meio físico escolhido para a transmissão. Por exemplo, pulsos de luz na fibra ótica, pulsos eletromagnéticos no cabo de cobre e diferentes modulações e faixas de frequências na transmissão sem fio.

### **2.1.3 Os modelos de referência OSI, TCP/IP e internet**

Após detalhada a pilha de protocolos chamada de Internet, é importante mencionar que este não é o único modelo em camadas de rede. Particularmente, no final da década de 70, a Organização Internacional para Padronização (ISO, *International Organization for Standardization*) propôs que as redes de computadores fossem organizadas em sete camadas, surgindo assim o modelo de Interconexão de Sistemas Abertos (OSI, *Open Systems Interconnection*). Na época muitos cursos universitários e de treinamentos obtiveram conhecimento deste padrão OSI e organizaram cursos voltados para o modelo em sete camadas. Por conta deste impacto precoce na educação de redes, o modelo de sete camadas continua presente em alguns livros sobre redes e em cursos de treinamento.

As sete camadas do modelo OSI, como mostradas na Figura 5 são: camada de aplicação, camada de apresentação, camada de sessão, camada de transporte, camada de rede,

camada de enlace e camada física. Com exceção de duas camadas, as restantes possuem as mesmas funções já descritas na subseção anterior. Estas duas camadas a mais neste caso, de apresentação e de sessão são detalhadas a seguir.

**Figura 5** – Modelo de referência OSI, TCP/IP e Internet.

Modelo OSI	Modelo TCP/IP	Modelo Internet
Aplicação	Aplicação	Aplicação
Apresentação		
Sessão		
Transporte	Transporte	Transporte
Rede	Rede	Internet
Enlace	Enlace	Interface com a rede
Física	Física	

Fonte: Elaboração do autor.

A camada de apresentação tem o papel de prover serviços que permitam que as aplicações de comunicação interpretem o significado dos dados trocados. Entre esses serviços estão a compressão de dados, a codificação de dados e a descrição de dados.

A camada de sessão provê a delimitação e sincronização da troca de dados, incluindo os meios de construir um esquema de pontos de verificação e de recuperação.

O fato do modelo TCP/IP ser desprovido destas duas camadas adicionais do modelo OSI faz surgir dois questionamentos: o primeiro, se os serviços oferecidos por estas camadas são irrelevantes, e o segundo, o que acontece se uma aplicação precisar de um desses serviços providos por alguma delas? A resposta está no propósito da própria aplicação, se o desenvolvedor julgar necessário algum destes serviços, cabe a ele mesmo programar. Desta forma, percebe-se que a camada de aplicação do modelo TCP/IP engloba e realiza sempre que necessário os serviços das duas camadas adicionais do modelo OSI, podendo se dizer que as camadas de apresentação e de sessão estão implícitas na camada de aplicação.

Outro modelo também referenciado em alguns livros didáticos de rede é o modelo Internet. Conforme ilustrado na Figura 5, este modelo possui apenas quatro camadas e é muito parecido com os outros dois modelos. As camadas de apresentação e de sessão também são

englobadas na camada de aplicação, ficando a diferença apenas em relação à cumulatividade das funções desempenhadas da camada de enlace e física para uma camada chamada de interface com a rede.

É importante ressaltar que embora existam diferentes modelos para se estruturar as camadas das redes de computadores, o fundamental é entender os conceitos que norteiam essas divisões em camadas. Cada uma destas camadas realiza um determinado serviço e disponibiliza interfaces para se comunicar com as camadas adjacentes, o que denota a existência de uma confiança por parte de cada uma delas no propósito e realização dos serviços das outras.

De uma forma geral, esta é a estrutura da Internet atual, com seus diversos componentes e funções, e um amadurecimento tamanho que proporciona comunicações intercontinentais em apenas segundos.

## 2.2 REDES DEFINIDAS POR SOFTWARE (*Software Defined Networks* - SDNs)

Esta mesma estrutura de rede que atualmente se encontra em pleno funcionamento e que se denomina Internet, apesar de bastante amadurecida, traz consigo alguns problemas que preocupam toda a comunidade científica de pesquisas em redes de comunicação.

Um desses maiores problemas é a falta de flexibilidade e escalabilidade. Alterações nos protocolos bem como implementações de novas tecnologias sempre dependem de alteração de hardware (roteadores e *switches*), o que significa alto custo financeiro e esforço excessivo dos administradores de redes.

Outro problema é o da falta de padronização e alta complexidade dos protocolos dos equipamentos de diferentes fabricantes. Como cada um produz seus roteadores e protegem seus protocolos, a Internet fica constituída de um número elevado de diferentes tecnologias, dificultando o trabalho dos administradores das redes.

Para tentar contornar esses problemas, a comunidade de pesquisa em redes de computadores tem investido em iniciativas que levem à implantação de redes com maiores recursos de programação, de forma que novas tecnologias e funcionalidades possam ser inseridas na rede de forma gradual e sem grande impacto financeiro. Exemplos de iniciativas desse tipo são as propostas de redes ativas (*active networks*) (TENNENHOUSE; WETHERALL, 2007), de *testbeds* como o PlanetLab (PETERSON; ROSCOE, 2006) e, mais recentemente, projeto GENI (TURNER, 2006; ELLIOTT; FALK, 2009). Redes ativas, apesar

de seu potencial, tiveram pouca aceitação pela necessidade de alteração dos elementos de rede para permitir que se tornassem programáveis (GUEDES et al., 2012).

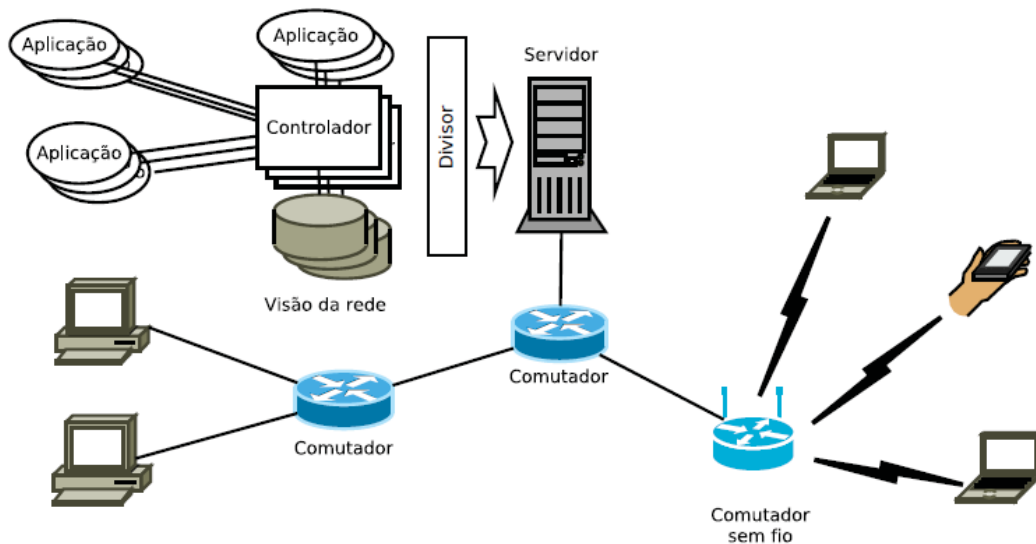
É também dentro deste contexto que surgiu o novo paradigma chamado Redes Definidas por Software (*Software Defined Network - SDN*). Uma estrutura que tem o objetivo de garantir o desempenho atual alcançado, no repasse e roteamento dos pacotes de dados, pois preserva a atual estrutura com os roteadores dedicados fazendo seu trabalho de retransmissão, mas que ao mesmo tempo delega a política de como isso será feito para um novo componente, chamado de **controlador**.

### 2.2.1 Os elementos das SDNs

A arquitetura de rede SDN é formada por três elementos: controladores, elementos de comutação programáveis e o protocolo padrão de comunicação entre os controladores e os elementos de rede.

Na Figura 6 apresenta-se a ilustração da estrutura geral de uma rede SDN e seus elementos.

**Figura 6** – Estrutura geral de uma SDN e seus componentes.



Fonte: Guedes et al. (2012).

Os controladores, também conhecidos como sistemas operacionais de rede ou *network hypervisors*, oferecem um ambiente de programação onde o desenvolvedor ou administrador pode ter acesso aos eventos gerados por uma interface de rede que siga um padrão como o

*OpenFlow* e podem também gerar comandos para controlar a infraestrutura de repasse e roteamento dos pacotes.

Este elemento concentra toda a comunicação com os elementos de comutação programáveis que compõem a rede, oferecendo assim uma visão unificada do estado da rede. Isto também simplifica a atividade dos administradores de rede, que não precisaram conhecer profundamente os detalhes da programação dos elementos de comutação.

Esta nova estrutura de controle da rede permite um gerenciamento mais efetivo e independente. Os controladores podem implementar lógicas de monitoramento do tráfego mais sofisticadas, por exemplo, uma solução que ofereça novas abstrações para os usuários, dando a cada um, a visão de que suas máquinas estão ligadas a um *switch* único e privado, independente dos demais.

Os comutadores e roteadores de rede, anteriormente independentes e autônomos, passam a ser configurados pelos controladores de rede que podem implementar as políticas de repasse, roteamento e de segurança baseadas em níveis de abstração maiores que o modelo atual pois foram definidas anteriormente pelos administradores de rede nos controladores.

Do ponto de vista histórico, SDNs têm sua origem na definição da arquitetura de redes *Ethane*, que definia um padrão de se implementar políticas de controle de acesso de forma distribuída, a partir de um mecanismo de supervisão centralizado (CASADO et al., 2009). Nesta arquitetura, cada elemento de rede deveria consultar o elemento supervisor ao identificar um novo fluxo. O supervisor consultaria um grupo de políticas globais para decidir, com base nas características de cada fluxo, como o elemento de encaminhamento deveria tratá-lo. Essa decisão seria comunicada ao comutador na forma da programação de uma entrada em sua tabela de encaminhamento com uma regra adequada para o novo fluxo (que poderia, inclusive, ser seu descarte). Esse modelo foi posteriormente formalizado por alguns dos autores na forma da arquitetura *OpenFlow* (GUEDES et al., 2012). Este é o terceiro e não menos importante elemento da arquitetura de redes SDNs. O protocolo de comunicação entre o controlador e os elementos de comutação programáveis, chamado *OpenFlow* está na linha de frente de SDN, tanto em relação ao mercado quanto a pesquisa.

O *OpenFlow* é um padrão aberto SDNs que tem como principal objetivo permitir que se utilize equipamentos de rede comerciais para pesquisa e experimentação de novos protocolos de rede, em paralelo com a operação normal das redes. Isso é conseguido com a definição de uma interface de programação que permite ao desenvolvedor controlar diretamente os elementos de encaminhamento de pacotes presentes no dispositivo. Com o *OpenFlow*, pesquisadores podem utilizar equipamentos de rede comerciais, que normalmente



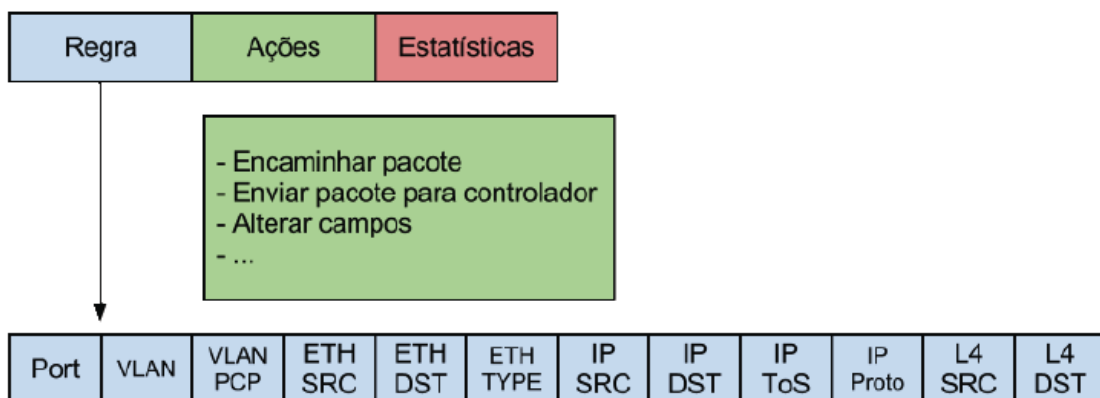
possuem maior poder de processamento que os comutadores utilizados em laboratórios de pesquisa, para realizar experimentos em redes “de produção”. Isso facilita a transferência dos resultados de pesquisa para a indústria (GUEDES et al., 2012).

### 2.2.2 Estrutura e comunicação entre os elementos

Um grande trunfo da arquitetura *OpenFlow* é a flexibilidade que ela oferece para se programar de forma independente o tratamento de cada fluxo observado, do ponto de vista de como o mesmo deve (ou não) ser encaminhado pela rede. Basicamente, o padrão *OpenFlow* determina como um fluxo de pacotes será tratado nos comutadores. Dentre as ações que podem ser realizadas para cada pacote estão o descarte ou repasse dos mesmos. O *OpenFlow* também define as regras de comunicação entre comutador e controlador, sendo utilizado para realizar alterações das regras ou até mesmo para contabilizar os fluxos. A união de uma definição de fluxo e um conjunto de ações forma uma entrada da tabela de fluxos *OpenFlow* (MCKEOWN et al., 2008).

Um *switch OpenFlow* permite que entradas diversas com regras de encaminhamento possam ser programadas em sua tabela. Um exemplo de entrada pode ser vista na Figura 7. Como o padrão é programado a partir do plano de controle, fluxos poderão ser definidos da forma escolhida pelo elemento controlador. Por exemplo, todos os pacotes enviados a partir do endereço físico A para o endereço físico B, ou todos os pacotes TCP enviados da máquina com endereço IP X para a porta 80 da máquina com endereço IP Y.

**Figura 7** – Exemplo de uma entrada na tabela de fluxos *OpenFlow*.



Fonte: Guedes et al. (2012).

Quando em operação, o comutador *OpenFlow*, analisa cada pacote que chega em quaisquer de suas interfaces de comunicação, ele compara as entradas da tabela de fluxos, anteriormente programada pelo controlador, com o cabeçalho do pacote; caso um casamento seja encontrado, considera-se que o pacote pertence aquele fluxo e então são aplicadas as ações relacionadas; se por acaso nenhuma regra for encontrada para o pacote, ele é encaminhado para o controlador para ser processado. Este processamento no controlador, para o fluxo não encontrado, pode resultar na criação de uma nova entrada no comutador para aquele fluxo, fazendo assim com que os próximos pacotes, contendo o mesmo cabeçalho, sejam encaminhados pelo próprio comutador. Isto evita o reenvio destes pacotes para o controlador e resulta em ganho de desempenho.

Esse procedimento cria diversas possibilidades, pois muitas das funcionalidades que são implementadas separadamente podem ser agrupadas em um único controlador *OpenFlow*, utilizando um pequeno conjunto de regras. Alguns exemplos das possibilidades são apresentados na Figura 8. As entradas representam o uso do *switch OpenFlow* para realizar encaminhamento de pacotes na camada de enlace, implementar um *firewall* e realizar encaminhamento de pacotes na camada de enlace utilizando *VLANs*, respectivamente (GUEDES et al., 2012).

**Figura 8** – Exemplos de uso de um *Switch OpenFlow*.

Port	VLAN	ETH SRC	ETH DST	IP SRC	IP DST	IP Proto	L4 SRC	L4 DST	Ações
*	*	*	00:4F:...	*	*	*	*	*	<b>Porto 4</b>
*	*	*	*	*	*	TCP	*	22	<b>DROP</b>
*	1	*	00:4F:...	*	*	*	*	*	<b>Porto 4, 6, 9</b>

Fonte: Guedes et al. (2012).

Um comutador de rede clássico (roteador ou *switch*), como visto no capítulo 2 deste trabalho, realiza o encaminhamento de pacotes rápido (*data path*) e as decisões de roteamento de alto nível (*control path*) no mesmo dispositivo. Neste paradigma SDN, um *switch OpenFlow* separa estas duas funções. A função de encaminhamento é realizada no *switch*, enquanto que as decisões de roteamento de alto nível são executadas em um controlador. Os *switches OpenFlow* e o controlador se comunicam através do protocolo *OpenFlow*, através de

conexões criptografadas, permitindo que todas mensagens como: *packet-received*, *send-packet-out*, *modify-forwarding-table*, e outras sejam trocadas entre os dois elementos de forma segura (MARCONDES, 2013).

Existem três ações básicas associadas a cada entrada na tabela de fluxos de um comutador *OpenFlow*, são elas:

- **Encaminhar os pacotes deste fluxo para uma determinada porta (ou portas).** Isso permite que os pacotes sejam roteados através da rede. Na maioria dos *switches* essa ação acontece na velocidade do hardware.
- **Encapsular e transmitir pacotes deste fluxo para um controlador.** Normalmente, isso é utilizado para o primeiro pacote de um novo fluxo, de modo que um controlador possa decidir se o fluxo deverá ser adicionado à tabela de fluxo do *switch*. Esta transferência entre o comutador e o controlador é feita através de um canal seguro (*secure channel*).
- **Descartar pacotes deste fluxo.** Isso pode ser usado por motivos de segurança, para conter ataques de negação de serviço (DoS - *Denial of Service*), ou para reduzir a transmissão de mensagens espúrias de descoberta enviadas a partir de *hosts*.

Cada entrada na tabela de fluxo possui três campos:

1. Uma regra que define o fluxo, isto é, como os pacotes são classificados como parte do fluxo. A regra consiste principalmente em fazer o *match* (correspondência) dos campos de cabeçalho do pacote;
2. A ação, que define como os pacotes devem ser processados; e
3. Estatísticas, que contabilizam o número de pacotes e bytes para cada fluxo, e o tempo decorrido desde o último pacote que fez correspondência com o fluxo (para ajudar com a remoção de fluxos inativos).

Alguns exemplos de entradas da tabela de fluxo e suas ações são mostrados na Figura 9. No primeiro exemplo, todos os pacotes cujos valores do campo (MAC dst) coincidirem, o comutador irá executar a ação de repasse para a porta número seis (port6). No exemplo do meio, foram especificados todos os campos da entrada, assim os pacotes que tiverem todos estes atributos coincidentes serão repassados também para (port6). No último exemplo, é especificada a regra de *firewall*, em que todos os pacotes que chegarem ao comutador e tiverem por destino a porta TCP 22 serão descartados.

Figura 9 – Exemplos de entradas e ações na tabela de fluxos.

### Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

### Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:2e:..	00:1f:..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

### Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Forward
*	*	*	*	*	*	*	*	*	22	drop

Fonte: Marcondes (2013).

Para processar os pacotes no *OpenFlow*, a maneira mais simples é forçar todos os pacotes de um fluxo (ou cabeçalhos de pacotes na tupla-10) a passarem pelo controlador. Para fazer isso, um controlador não precisa adicionar novas entradas de fluxos no *switch OpenFlow* – ele só deixa o *switch* redirecionar por padrão todos os pacotes para o controlador. Isto tem a vantagem da flexibilidade em detrimento do desempenho e pode proporcionar uma maneira útil para testar a funcionalidade de um novo protocolo, mas é pouco provável que seja de muito interesse de implantação em grande rede. Alternativamente, pode-se definir uma entrada na tabela de fluxo do *switch*, assim que o controlador decidir sobre a ação correspondente (baseado no protocolo que vamos implementar). Quando pacotes subsequentes chegarem ao *switch*, eles serão rapidamente processados na velocidade máxima do hardware pela tabela de fluxos (MARCONDES, 2013).

## 2.3 FERRAMENTAS PARA SIMULAÇÃO E TESTES SDNs

Nesta seção serão demonstradas ferramentas para prototipagem e simulação de redes definidas por software. Serão apresentados também resultados de alguns testes realizados para avaliar essas ferramentas.

### 2.3.1 O emulador MININET

Por ser o paradigma de SDN um tema muito recente, boa parte da comunidade acadêmica de pesquisa em redes tem se voltado para os estudos do tema. Estes pesquisadores quando desejam testar novas funcionalidades em SDN, nos controladores, nos comutadores ou mesmo no protocolo *OpenFlow*, encontram grande dificuldade, pois ainda existem poucos equipamentos que implementam os padrões SDN a custo baixo e em casos mais específicos, quando é necessário simular grandes redes com grandes quantidades de *hosts*, *switches* e *controladores SDNs*, utilizar a própria Internet pode não ser uma boa ideia, pois configurações mal sucedidas podem prejudicar seu funcionamento.

Uma das soluções para este problema é a criação de protótipos e a simulação destes ambientes de forma virtual. Neste sentido algumas ferramentas têm sido criadas e uma delas é o software Mininet (LANTZ; HELLER; MCKEOWN, 2010).

O Mininet é um sistema que permite a rápida prototipação de grandes redes utilizando apenas um computador. Ele cria SDNs escaláveis utilizando primitivas de virtualização do sistema operacional, como processos e *namespaces* de rede. Com essas primitivas, ele permite rapidamente criar, interagir e customizar protótipos de SDNs (GUEDES et al., 2012).

Experiências de implementações sugeriram que a capacidade de executar, criar e avaliar SDNs em tempo real também representa uma mudança qualitativa no fluxo de trabalho. Estudos de casos selecionados de mais de 100 usuários, em 18 instituições, que pesquisam SDNs, mostraram níveis de desenvolvimentos acentuados a partir da rede colaborativa criada com protótipos SDN. Nesta rede colaborativa, qualquer pesquisador pode executar, avaliar, explorar e construir em cima de protótipos já estudados por outros pesquisadores (LANTZ; HELLER; MCKEOWN, 2010).

Alguns atributos nortearam a criação do Mininet, são eles:

- **Flexibilidade:** novas topologias e novas funcionalidades podem ser definidas em software, usando linguagens de programação e sistemas operacionais familiares;

- **Aplicabilidade:** implementações em protótipos feitos de forma correta devem ser utilizáveis também em redes reais baseadas em hardware sem nenhuma alteração nos códigos;
- **Interatividade:** o gerenciamento e o funcionamento da rede simulada devem ocorrer em tempo real, como acontece em redes reais;
- **Escalabilidade:** O ambiente de prototipagem deve ser escalável para redes de centenas ou milhares de *switches* em um simples microcomputador;
- **Realidade:** O comportamento do protótipo criado deve representar alto grau de similaridade com ambientes reais. Por exemplo, aplicações e protocolos devem ser utilizados tanto no ambiente real quanto no simulado, sem nenhuma modificação de código e gerar comportamentos e resultados iguais;
- **Compartilhamento:** Protótipos criados podem ser facilmente compartilhados com outros colaboradores que poderão executar, modificar e testar os experimentos.

### 2.3.1.1 Elementos de SDN no Mininet

O Mininet pode criar elementos de SDN, customizá-los, compartilhá-los com outras redes e executar interações. Tais elementos incluem:

- **Hosts:** Um *host* no Mininet é simplesmente um processo sendo executado no sistema operacional com seu próprio ambiente de rede. Cada um possui sua interface de rede virtual, suas portas, endereços e tabelas de roteamento ARP e IP;
- **Switches:** Os *switches OpenFlow* criados pelo Mininet, possuem a mesma semântica de entrega de pacotes provida por um *switch* físico. Espaços de usuário e de *kernel* são disponibilizados;
- **Controladores:** Para simulação no Mininet, os controladores podem ser tanto simulados quanto reais, desde que as máquinas onde os *switches* estão sendo executados possuam conectividade com o controlador. O Mininet, se desejado, cria um controlador padrão dentro do próprio ambiente local da simulação;
- **Links:** Conexões ethernet virtuais são criadas entre os elementos através de suas interfaces virtuais.

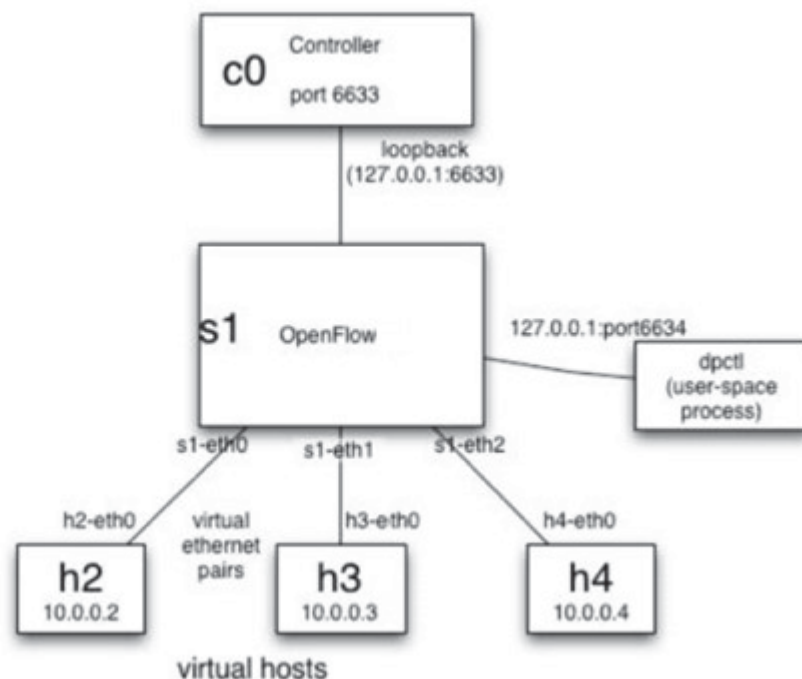
Por exemplo, o comando `mn -topo single,3 -mac -switch ovsk -controller remote`, executa as seguintes tarefas:

- Criação de três *hosts* virtuais, cada um contendo um endereço IP diferente;
- Criação de um único *switch* em software *OpenFlow* no *kernel* com 3 portas;
- Conexão de cada *host* virtual criado ao *switch* com um cabo ethernet virtual;
- Definição do endereço MAC parecido com seu endereço IP para cada *host*;
- Configuração do *switch OpenFlow* para se conectar a um controlador remoto.

Neste caso o controlador está sendo executado localmente no mesmo hardware que está rodando o simulador Mininet.

Na Figura 10 ilustra-se este protótipo de rede SDN no Mininet.

**Figura 10** – Exemplo de protótipo SDN no Mininet.



Fonte: Marcondes (2013).

### 2.3.1.2 Interatividade com a rede do protótipo e ferramentas de controle

Depois de criados os elementos e todas as conexões entre eles, é desejável que se consiga executar comandos nos *hosts* para testar as funcionalidades da rede, verificar o funcionamento dos *switches* e dos controladores.

Para isso o Mininet inclui uma interface de linha de comando que permite aos desenvolvedores controlar e gerenciar a rede. Os comandos digitados são interpretados pelo

emulador e executados no ambiente de rede simulado, como por exemplo, o comando “*h2 ping h3*” que executa um *ICMP echo request* do *host h2* para o *host h3*. Dentro do ambiente de simulação, o pacote em questão será criado no *host h2*, será enviado para o *switch s1*, onde será tratado pela tabela de fluxos e encaminhado para o *host h3*.

Outros exemplos de comandos que podem ser executados pelo emulador são:

`mininet > nodes` - Visualiza a lista de nós disponíveis;

`mininet > help` - Visualiza a lista de comandos disponíveis para a rede;

`mininet > h2 ifconfig` - Verifica o endereço IP de um determinado *host*, o *h2* neste caso.

Caso se queira customizar uma rede, o emulador Mininet disponibiliza uma interface de programação (API) na linguagem Python, onde é possível criar experimentos, topologias e diferentes tipos de nós: *switches*, controladores, *hosts* e outros. Algumas linhas de código Python são suficientes para se definir um cenário customizado que cria uma rede, executa comandos em múltiplos nós e mostra os resultados (LANTZ; HELLER; MCKEOWN, 2010).

Por exemplo, o código a seguir cria uma pequena rede de quatro *hosts* e três *switches*, e realiza um *ping* do *host 1* para o *host 4*.

```
from mininet.net import Mininet
from mininet.topolib import TreeTopo
tree4 = TreeTopo(depth=2,fanout=2)
net = Mininet(topo=tree4)
net.start()
h1, h4 = net.hosts[0], net.hosts[3]
print h1.cmd('ping -c1 %s' % h4.IP())
net.stop()
```

Diversos exemplos e ferramentas, incluindo *scripts* baseados em texto e aplicações gráficas, estão inclusas na máquina virtual que contém o *Mininet*.

Duas destas ferramentas são mostradas nas Figuras 11 e 12, a primeira mostra o monitoramento dos múltiplos *hosts*, *switches* e controladores e a segunda mostra a topologia da rede criada com seus nós e links.



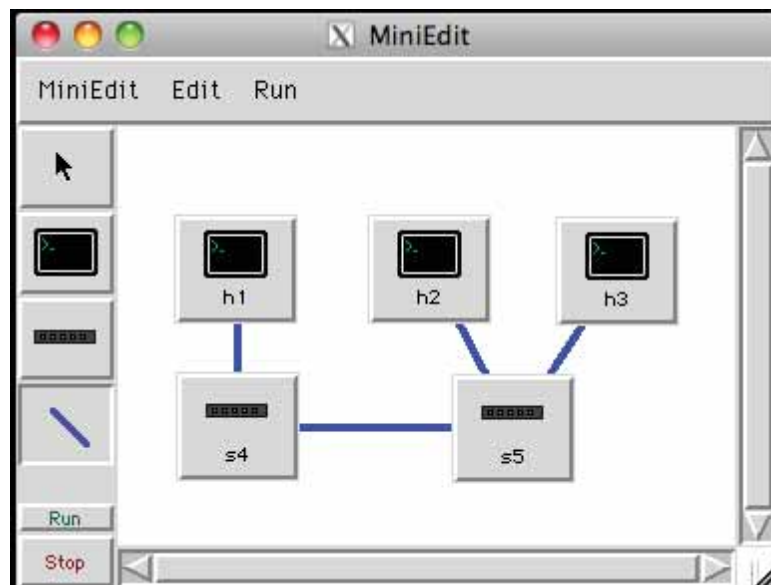
Figura 11 – Script console.py monitorando os *hosts* da rede.



Hosts	Switches	Controllers	Graph	Ping	Iperf	Interrupt	Clear
h1							
15.6 MBytes Mbits/sec	131	14.5 MBytes Mbits/sec	122	13.7 MBytes Mbits/sec	115	11.7 MBytes Mbits/sec	98.1
h2							
h3							
h4							
h5							
13.7 MBytes Mbits/sec	115	14.4 MBytes Mbits/sec	121	15.0 MBytes Mbits/sec	126	12.2 MBytes Mbits/sec	103
h6							
h7							
h8							
h9							
14.8 MBytes Mbits/sec	124	13.9 MBytes Mbits/sec	116	15.3 MBytes Mbits/sec	129	10.8 MBytes Mbits/sec	90.4
h10							
h11							
h12							
h13							
14.5 MBytes Mbits/sec	121	16.8 MBytes Mbits/sec	141	14.5 MBytes Mbits/sec	122	12.2 MBytes Mbits/sec	103
h14							
h15							
h16							

Fonte: Lantz, Heller e Mckeown (2010).

Figura 12 – Ferramenta MiniEdit utilizada para editar a topologia da rede.



Fonte: Lantz, Heller e Mckeown (2010).

Outras duas ferramentas muito úteis para controlar e averiguar o funcionamento das redes emuladas pelo Mininet são o *dpctl* e o aplicativo *wireshark*.

O *dpctl* é um utilitário que permite visibilidade e controle sobre uma tabela de fluxos de um único *switch*. Ele é especialmente útil para *debugging*, pois permite exibir o estado dos

fluxos e dos contadores de fluxo. A maioria dos *switches OpenFlow* inicia seus serviços com uma porta de escuta passiva (6634 por padrão), na qual é possível interagir com o *switch*, sem ter que adicionar código de *debugging* no controlador.

Por exemplo, o comando `dpctl dump-flows tcp:127.0.0.1:6634`, conecta-se ao *switch* e mostra como resultado a tabela de fluxos instalada.

É também possível inserir manualmente fluxos nas tabelas dos *switches*. O comando `dpctl add-flow tcp:127.0.0.1:6634 in_port=1, actions=output:2`, por exemplo, cria uma regra em que todos pacotes que chegarem a porta 1 do *switch* serão encaminhados para a porta 2.

*Wireshark* é uma ferramenta que captura e dissectiona todos os pacotes de dados transmitidos por determinadas interfaces de rede. Exclusivamente para o Mininet, o *wireshark* possui uma biblioteca *OpenFlow(of)* que filtra todos os pacotes do tipo *OpenFlow*, isto permite que os pacotes *OpenFlow* capturados pela interface local do hardware que está executando o emulador, sejam separados dos outros pacotes de outras aplicações. Para se utilizar a ferramenta, basta executá-la junto a um servidor gráfico em execução.

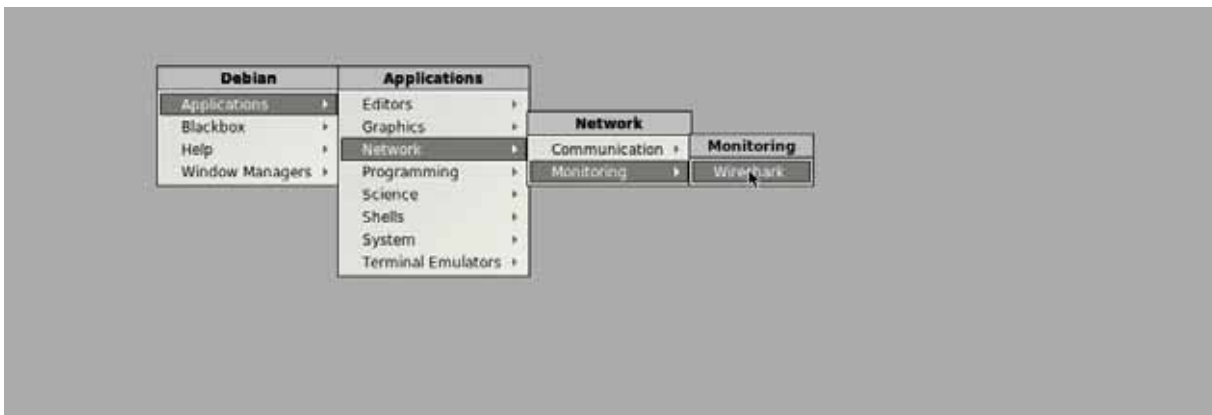
O comando `startx` no console da máquina virtual do Mininet executa o servidor gráfico como ilustrado na Figura 13, após isto é necessário abrir o *menu* e selecionar a ferramenta *wireshark*, como mostrado na Figura 14.

**Figura 13** – Comando para iniciar console gráfico.

```
root@mininet-vm:/home/mininet# startx
```

Fonte: Elaboração do autor.

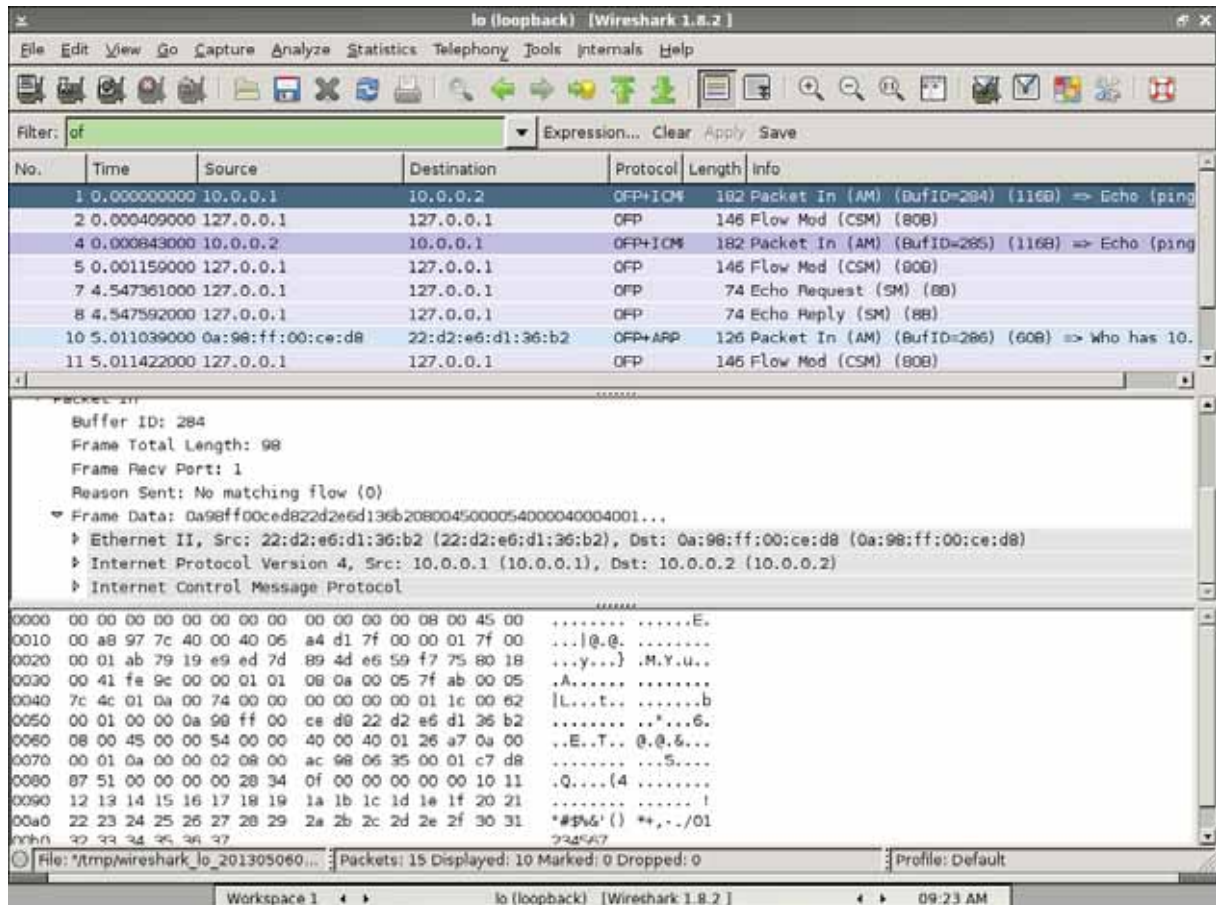
**Figura 14** – Executar a ferramenta *wireshark* no console gráfico.



Fonte: Elaboração do autor.

A Figura 15 mostra a interface da ferramenta *wireshark* em execução e alguns pacotes capturados por ela.

**Figura 15** – Ferramenta *wireshark* em execução.



Fonte: Elaboração do autor.

### 2.3.1.3 O emulador e os controladores SDNs

O princípio por trás de Redes Definidas por Software é a capacidade de se controlar o plano de encaminhamento de pacotes através de uma interface bem definida. O controlador pode concentrar a comunicação com todos os elementos programáveis que compõem a rede e fornecer uma visão unificada do estado da rede isolando os detalhes de cada elemento.

Um dos pontos fortes de redes SDNs é exatamente a formação dessa visão centralizada das condições da rede, sobre a qual é possível desenvolver análises detalhadas e chegar a decisões operacionais sobre como o sistema como um todo deve operar (GUEDES et al., 2012).

O emulador Mininet possibilita a conexão dos *switches* criados na rede a diversos controladores, como POX, NOX e *Floodlight* por exemplo. Esta possibilidade permite que desenvolvedores interessados em testar e criar recursos de programação nos controladores possam utilizar o Mininet para realizar as simulações de fluxos de informações.

O NOX é o controlador original *OpenFlow* e tem como principal função hoje o desenvolvimento de controladores eficientes em linguagem de programação C++. Ele opera sobre o conceito de fluxos de dados. Ele verifica o primeiro pacote de cada fluxo e procura na tabela de fluxos para determinar a política a ser aplicada. O controlador gerencia o conjunto de regras instaladas nos *switches* da rede reagindo a eventos de rede. Atualmente a maioria dos projetos de pesquisa na área é baseada no controlador NOX, que possui um sistema operacional simples para redes e que provê primitivas para o gerenciamento de eventos bem como as funções para a comunicação com os *switches* (GUDE et al., 2008).

O controlador NOX obteve uma grande aceitação entre os pesquisadores da área de SDN. A existência de duas interfaces, C++ e Python, permite que o mesmo ambiente seja utilizado em situações que exigem alto desempenho e em casos onde a capacidade de expressão de Python agiliza o desenvolvimento e simplifica o código resultante.

Segue um exemplo da programação no controlador NOX:

```
def switch_join(switch):
    repeater(switch)

def repeater(switch):
    pat1 = {in_port:1}
    pat2 = {in_port:2}

install(switch,pat1,DEFAULT,None,[output(2)])
install(switch,pat2,DEFAULT,None,[output(1)])
```

Neste exemplo, quando um *switch* se conecta à rede, o controlador instala regras no *switch* instruindo-o a enviar pacotes da porta 1 para a porta 2 e vice-versa. As regras são permanentes, com prioridade padrão (*default*) e tempo sem limite (*None*).

O *Floodlight* é um controlador *OpenFlow* para redes corporativas baseado totalmente na linguagem Java e distribuído segundo a licença Apache. O projeto se originou do controlador *Beacon* e agora é apoiado por uma comunidade de desenvolvedores e também pela *Big Switch Networks*, *start-up* que produz controladores comerciais que suportam o

*Floodlight*. O núcleo e módulos principais são escritos em Java. Recentemente adicionaram *Jython*, o que permite desenvolvimento na linguagem Python. Em sua arquitetura, todos os elementos são módulos e módulos exportam serviços. Toda a comunicação entre módulos é feita através de serviços. A interface *ItopologyService* permite descobrir a topologia da rede automaticamente. Permite integrar com redes não *OpenFlow* e é compatível com a ferramenta de simulação Mininet.

Desenvolvido especificamente para o ensino e pesquisa, o controlador POX é o irmão mais novo do NOX. Sua essência é uma plataforma para o desenvolvimento e a prototipagem rápida de aplicações de software para redes SDN usando a linguagem de programação Python.

O POX está em constante desenvolvimento e têm o objetivo de substituir o NOX. Sua arquitetura, baseada no NOX, é mais estável e sua interface é mais elegante resultando em um controlador mais moderno e simples.

Embora o POX seja mais estável, o NOX ainda permanece como um ambiente adequado para implementações que exijam demandas mais elevadas em termos de desempenho. No entanto, os desenvolvedores do POX acreditam que este seja mais adequado para substituir o NOX nos casos em que Python é utilizado. Esse controlador foi escolhido para a programação do módulo proposto neste trabalho devido a sua ótima praticidade de uso e boa capacidade de respostas frente aos diversos testes executados.

#### **2.3.1.4 Simulando SDN com Mininet e POX**

Após conhecidas as principais ferramentas para simulação de SDNs, esta seção apresenta alguns cenários de utilização do software Mininet juntamente com software controlador POX. O objetivo é testar o comportamento da ferramenta de emulação Mininet em diferentes topologias de rede e obter um resultado de sua viabilidade.

#### **2.3.1.5 Especificação do ambiente de emulação**

Para a realização dos experimentos foi utilizado um microcomputador HP Compaq 8200 Elite SFF PC contendo as seguintes especificações:

- Processador Intel® Core™ i5-2400 CPU @ 3.10GHz;
- 4 Gb de Memória RAM;
- Sistema operacional Windows 7 64 bits;

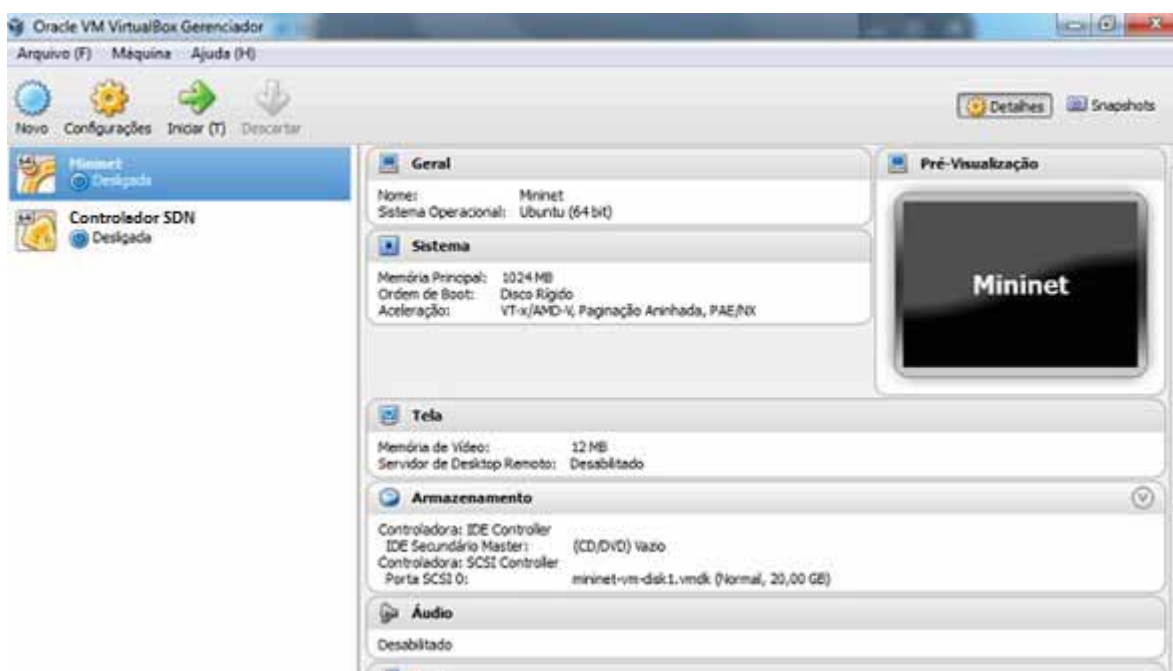
- Oracle VM VirtualBox versão 4.2.12

Neste microcomputador sob gerenciamento do VirtualBox, foram instalados os seguintes sistemas operacionais:

1. Máquina virtual com o software emulador Mininet versão 2.0:
  - Sistema operacional Linux Ubuntu 12.10 64 bits;
  - 1 Gb de Memória RAM;
2. Máquina virtual com o software controlador POX:
  - Sistema operacional Linux Ubuntu 12.10 64 bits
  - 256 Mb de memória RAM.

Na Figura 16 ilustra-se o Oracle VM VirtualBox com as duas máquinas virtuais instaladas.

**Figura 16** – Oracle VM com o Mininet e o POX instalados.

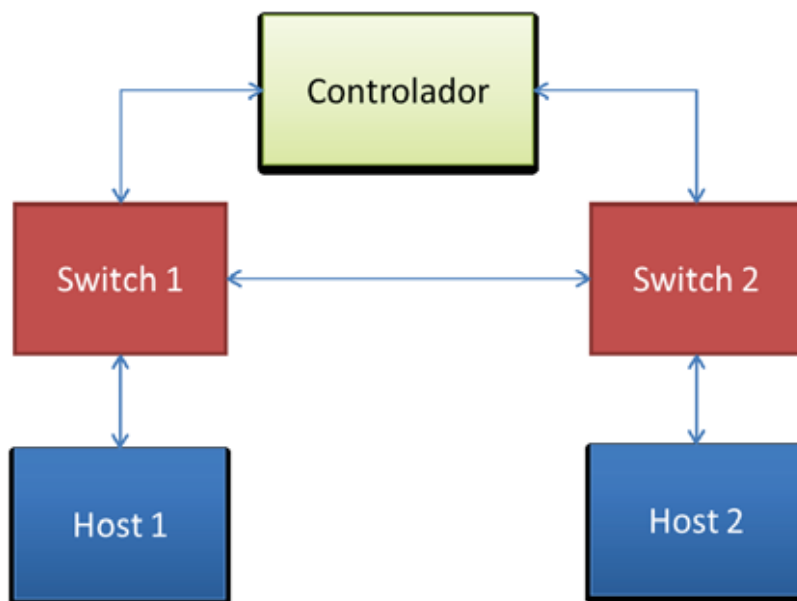


Fonte: Elaboração do autor.

### 2.3.2 Testes de funcionamento – 2 hosts, 2 switches e 1 controlador

Neste primeiro cenário de simulação, foram criados dois *hosts*, dois *switches*, um controlador e interligados os nós com *links* cabeados, conforme topologia ilustrada na Figura 17.

**Figura 17** – Topologia do cenário de simulação.



Fonte: Elaboração do autor.

O comando para a criação do cenário proposto no ambiente simulado foi o seguinte:

```
mn -topo=linear,2
```

Depois de criado o cenário, foi disparado um comando de teste de conexão, utilizando pacote ICMP (*echo-request* “ping”) entre os *hosts* 1 e 2, conforme comando a seguir:

```
h1 ping -c1 h2
```

O resultado destes testes pode ser visualizado na Figura 18. O Mininet adicionou o controlador, os *hosts*, os *switches* e em seguida criou os links entre todos eles. Logo após, o comando ICMP de teste obteve êxito com 0% de perda do pacote. É importante enfatizar que nenhuma configuração adicional foi feita nos *switches* ou no controlador, portanto, a comunicação somente ocorreu com sucesso porque o pacote ICMP enviado do *host* 1 para *host* 2 ao passar pelo *switch* 1 que não possuía nenhuma regra de repasse em sua tabela de fluxo, encaminhou o pacote para o controlador, que por sua vez repassou para o *switch* 2.

Figura 18 – Comandos e resultados no cenário.

```
root@mininet-vm:/home/mininet# mn --topo=linear,2
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s1, s2)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 2 switches
s1 s2
*** Starting CLI:
mininet> h1 ping -c1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=5.68 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.680/5.680/5.680/0.000 ms
mininet> █
```

Fonte: Elaboração do autor.

Devido ao fato de nenhum dos dois *switches* possuírem suas respectivas tabelas de fluxos, o desempenho do repasse dos pacotes na rede é prejudicado, tendo em vista que todos os pacotes terão que passar pelo controlador antes de chegarem ao seu destino. Isto cria uma dependência pontual e um quadro de possível congestionamento no controlador de rede.

Para exemplificar este congestionamento, foram realizados mais alguns testes ainda neste cenário.

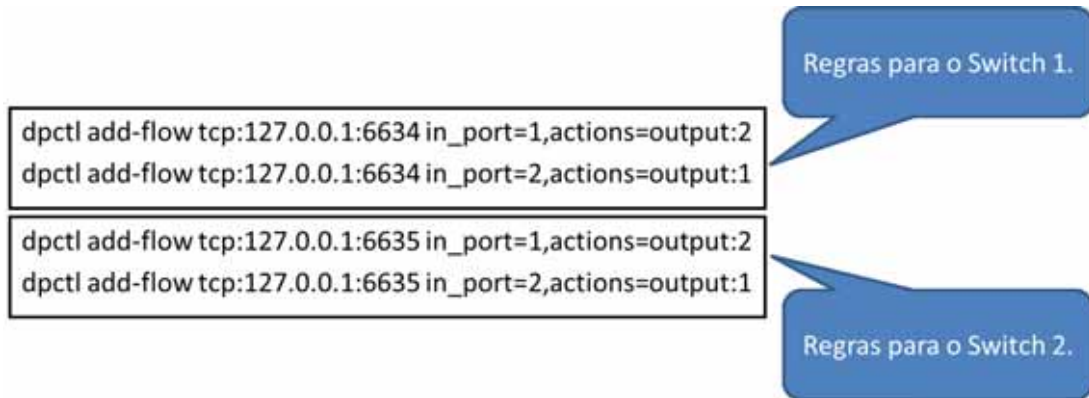
O controlador foi desconectado do ambiente SDN e o teste de comunicação com o pacote ICMP foi repetido entre o *host 1* e o *host 2*. Como resultado, a comunicação não pode ser estabelecida, gerando 100% de perda dos pacotes. Ocorreu que o *switch 1*, que não possuía nenhuma regra de repasse, ao receber o pacote ICMP, tentou enviá-lo ao controlador. Como isso não foi possível devido à ausência do controlador na rede, a comunicação foi malsucedida.

Posteriormente, ainda com o controlador desconectado, foram implantadas regras de repasse nas tabelas de fluxo nos dois *switches* e repetido o teste de comunicação com o pacote ICMP. O resultado foi positivo, o pacote ICMP conseguiu chegar até seu destino, pois com as regras implantadas nos *switches*, estes não precisaram da ajuda do controlador para realizar os



repasses. A Figura 19 contém as regras aplicadas nas tabelas dos *switches* para a realização do teste.

**Figura 19** – Regras aplicadas nas tabelas de fluxo dos *switches*.



Fonte: Elaboração do autor.

É importante considerar que estas regras de fluxos foram adicionadas aos *switches* de forma manual neste teste, conforme comandos a seguir, mas na arquitetura SDN, esta tarefa fica a cargo do controlador que pode incluir, modificar e apagar quaisquer regras.

### 2.3.2.1 Testes de escalabilidade

No mesmo ambiente foram realizados alguns testes de escalabilidade que serão explicados a seguir. No sistema operacional Linux Ubuntu, foi criado um *script bash* que inicializa uma determinada topologia de rede virtual no Mininet, imprime a quantidade de memória utilizada até aquele momento e depois finaliza a rede virtual imprimindo o tempo total gasto no procedimento (OLIVEIRA et al., 2014a; 2014b). O *script* mencionado está transcrito a seguir.

```
#!/bin/bash
```

```
for n in `seq 1 6`; do
```

```
    echo "n: $n"
```

```
    echo -e "h1 free -m | grep ^Mem | awk '{print \"mem: \" \$3}'\nexit" | \
```

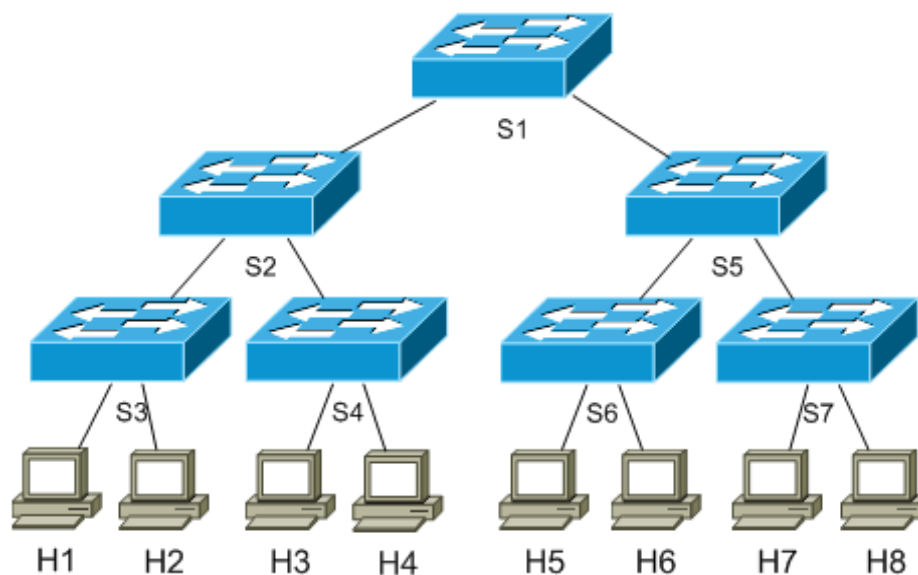
```
mn --topo tree,$n 2>&1 | \
```

```
grep -E "(mem:|completado em)"
```

```
done
```

Uma das topologias criadas pelo código acima pode ser vista na figura 20.

**Figura 20** – Topologia com 15 nós.



Fonte: Elaboração do autor.

Os resultados obtidos na execução do *script* estão ilustrados na Tabela 1.

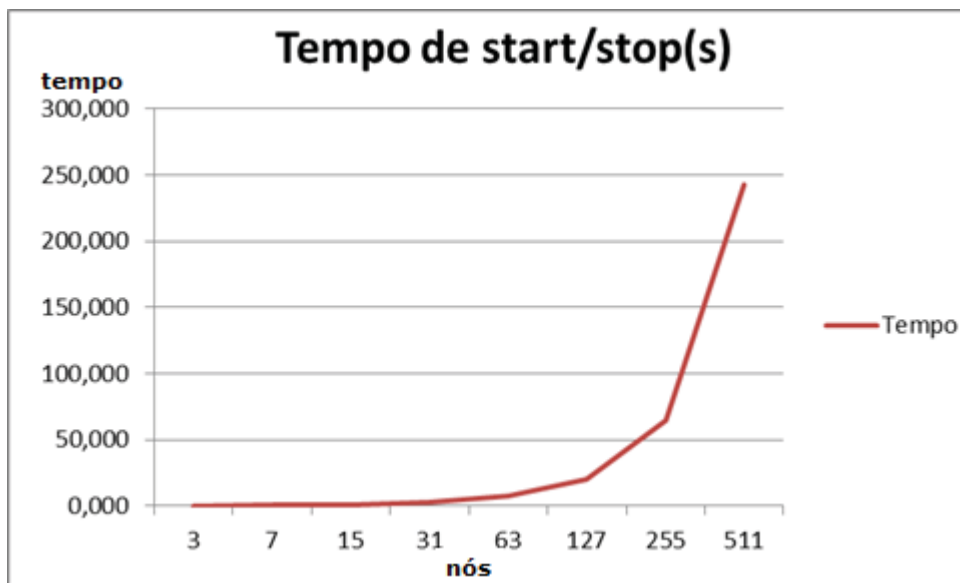
**Tabela 1** – Resultados obtidos nos testes de escalabilidade.

Topologia	Número de nós	Número de hosts	Número de switches	Tempo de start/stop(s)	Memória Ocupada(MB)
Tree	3	2	1	0,190	441
Tree	7	4	3	0,525	442
Tree	15	8	7	1,175	445
Tree	31	16	15	2,795	452
Tree	63	32	31	7,088	466
Tree	127	64	63	20,210	449
Tree	255	128	127	64,818	572
Tree	511	256	255	242,842	739

Fonte: Elaboração do autor.

Nota-se que para a criação de redes virtuais pequenas, o tempo gasto é também muito pequeno e este tempo vai crescendo de forma exponencial conforme o número de nós virtuais criados é aumentado. Por exemplo, o Mininet demorou aproximadamente 64 segundos para criar uma topologia em árvore com 255 nós, enquanto que com 511 nós o tempo gasto subiu consideravelmente para aproximadamente 242 segundos. No gráfico apresentado na Figura 21 ilustra-se este fato.

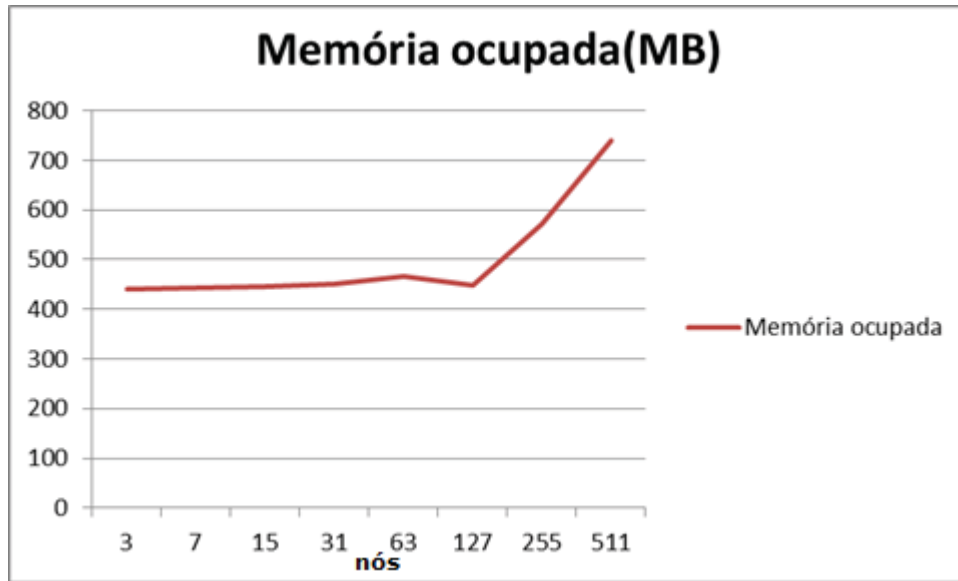
**Figura 21** – Gráfico de tempo gasto para criação e destruição das redes.



Fonte: Elaboração do autor.

Já em relação à quantidade de memória ocupada na criação das redes virtuais, conforme ilustrado no gráfico da Figura 22, o Mininet demonstrou crescimento menos acentuado no consumo de memória conforme o número de nós foi aumentando. Percebe-se que o consumo de memória foi praticamente o mesmo até a criação da rede virtual com 127 nós e após isso o consumo cresceu praticamente de forma linear.

**Figura 22** – Gráfico representando a memória ocupada para as redes.



Fonte: Elaboração do autor.

O trabalho divulgado por (LANTZ; HELLER; MCKEOWN, 2010), também realizou alguns testes de escalabilidade de forma semelhante aos realizados neste trabalho. Na Tabela 2 ilustra-se a tabela de resultados obtidos pelos experimentos, onde H representa o número de *hosts* e S o número de *switches*. Os testes foram realizados utilizando um MacBook Pro (2.4 GHz Intel Core 2 Duo com 6 GB de memória RAM), em um sistema operacional Debian 5/Linux 2.6.33.1 virtualizado com o gerenciador VMware Fusion 3.0.

**Tabela 2** – Resultados obtidos por outro trabalho similar.

Topology	H	S	Setup(s)	Stop(s)	Mem(MB)
Minimal	2	1	1.0	0.5	6
Linear(100)	100	100	70.7	70.0	112
VL2(4, 4)	80	10	31.7	14.9	73
FatTree(4)	16	20	17.2	22.3	66
FatTree(6)	54	45	54.3	56.3	102
Mesh(10, 10)	40	100	82.3	92.9	152
Tree(4 <sup>4</sup> )	256	85	168.4	83.9	233
Tree(16 <sup>2</sup> )	256	17	139.8	39.3	212
Tree(32 <sup>2</sup> )	1024	33	817.8	163.6	492

Fonte: Lantz, Heller e Mckeown (2010).

Excetuando-se apenas pelas diferenças encontradas no poder de processamento dos *hardwares* utilizados em ambos os testes, nota-se que os resultados obtidos foram semelhantes, o Mininet começa a demorar mais para criar e destruir as redes virtuais e

consumir recursos mais significativos de memória quando o número de nós das redes virtuais se torna maior.

### **2.3.2.2 Resultados e discussão**

Este novo paradigma de SDNs tem recebido grande atenção por parte da comunidade científica e de empresas desenvolvedoras de equipamentos e sistemas de gerência de redes. Um dos indicadores que provam isso foram os mais de dez trabalhos focados na área publicados no Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) de 2011. Outro evento chamado de *Open Networking Summit*, um encontro de pesquisadores e profissionais da indústria, organizado pela Universidade de Stanford e a empresa Nicira, teve um número de participantes muito acima do esperado pelos organizadores (GUEDES et al., 2012).

Por ser um tema atual, com tendências inovadoras e capaz de proporcionar diversas possibilidades de aplicação, este paradigma está sendo chamado de a “Internet do futuro”. Neste sentido, diversos fabricantes e pesquisadores aderiram à ideia de uma estrutura de redes definida por aplicações independentes de hardware e não mais pelo princípio apenas do roteamento IP tradicional. Alguns fabricantes como HP, Dell, Brocade, Datacom e outros já estão, inclusive, disponibilizando no mercado equipamentos que implementam a interface padronizada *OpenFlow*.

Neste sentido o *software* emulador Mininet abordado neste trabalho, traz consigo uma contribuição ímpar para o avanço dos estudos em SDN e após a realização de vários testes deste autor e a pesquisa intensa de estudos de casos de outros pesquisadores, seguem as conclusões.

### **2.3.2.3 Limitações**

A limitação mais significativa do Mininet atualmente é a falta de fidelidade no desempenho, especialmente em altas cargas. Recursos da CPU são multiplexados no tempo pelo escalonador padrão do sistema operacional *Linux*, e este não oferece nenhuma garantia de que um *host* que está pronto para enviar um pacote será agendado prontamente, ou que todos os *switches* irão encaminhar com a mesma taxa de transferência (LANTZ; HELLER; MCKEOWN, 2010).

O Mininet atualmente é executado em uma única máquina e emula todos os *hosts*, *switches* e *links* em um único sistema operacional, desta forma todos os nós compartilham os mesmos recursos de hardware, sendo esta uma desvantagem para experimentos em maior escala.

#### **2.3.2.4 Prototipação**

Apesar de não ser indicado para grandes simulações que exijam quantidades de nós significativos, o Mininet é um ótimo *software* para prototipação de redes pequenas e médias.

Estudantes, pesquisadores, administradores de redes ou outros interessados, com um simples *laptop* podem usar o Mininet para prototipação rápida de uma ideia de SDN. O baixo tempo de inicialização e a baixa sobrecarga facilitam a exploração de um espaço de projeto e construção de um sistema de escala interessante que pode ser executado em emulação em um hardware modesto. Vários pesquisadores podem compartilhar *scripts*, configurações, topologias e trabalhar simultaneamente sem interferência.

#### **2.3.2.5 Aplicabilidade**

Um dos requisitos mais importantes para qualquer *software* emulador é a aplicabilidade. Neste contexto, todos os recursos e ideias implementadas em ambiente de simulação devem funcionar adequadamente em ambientes reais sem necessidade de alteração considerável nos códigos.

O Mininet prova esta capacidade, podendo ser implantado em redes de pesquisa, de produção, de validação, medição e uso geral. Ele preserva toda a semântica de códigos e *scripts* de configuração entre a emulação e o real, permitindo que os testes desenvolvidos em um ambiente emulado possam ser facilmente compartilhados para outras infraestruturas de teste, independentes de hardware ou mesmo em ambiente de produção.

#### **2.3.2.6 Compartilhamento**

Este é talvez uns dos pontos mais fortes do emulador Mininet, a capacidade de compartilhar recursos, protótipos e ideias. Um projeto, uma topologia ou códigos de teste podem de forma muito fácil, serem distribuídos para toda a comunidade de pesquisa. O

próprio emulador, quando obtido através do site oficial “mininet.org”, inclui no mesmo pacote da máquina virtual, o Mininet pré-instalado, diversas ferramentas para executar e analisar protótipos SDN e um conjunto de códigos adicionais de exemplo, que criam diversas topologias de redes SDN e facilitam o desenvolvimento das pesquisas (LANTZ; HELLER; MCKEOWN, 2010).

É possível também que pesquisadores que criarem ferramentas adicionais, códigos de exemplos e diferentes configurações SDN, possam compartilhá-las com os outros pesquisadores. Basta que seja gerada uma nova imagem de sua máquina virtual e a mesma seja distribuída através da Internet.

### **2.3.3 Conclusão**

Esta seção teve por objetivo o estudo de simulações em SDNs.

Para a prototipação e simulação de redes SDNs, foi adotada a ferramenta Mininet e o controlador POX e também incluído o resultado de outros pesquisadores.

Depois de realizados diversos testes e simulações com as ferramentas adotadas, concluiu-se nesta seção que, apesar das limitações apresentadas em relação ao desempenho no ambiente simulado com um número significativamente grande de nós, a ferramenta Mininet tem grande valia no tocante a realização de pesquisas em SDNs. Sua capacidade de prototipagem rápida e simplificada, sua garantia de aplicabilidade em relação ao ambiente real e a possibilidade facilitada de compartilhamento de resultados e ferramentas, ajudam os pesquisadores da área a impulsionar seus estudos.

As seções a seguir deste trabalho estudam aspectos de segurança em redes definidas por software utilizando a ferramenta de simulação Mininet juntamente com o controlador POX.

### 3 O PROBLEMA DE SEGURANÇA DO PROTOCOLO DE RESOLUÇÃO DE ENDEREÇOS (ARP)

Nesta seção será explicado em detalhes o funcionamento do protocolo de resolução de endereços (ARP), suas vulnerabilidades frente ao ataque *Man-in-the-Middle* (MITM) tanto na estrutura atual das redes de computadores quanto no novo paradigma de redes definidas por software.

#### 3.1 O PROTOCOLO DE RESOLUÇÃO DE ENDEREÇOS (ARP)

Todos os dispositivos conectados a uma rede TCP/IP são identificados na camada de rede por seu endereço IP de 32 *bits* e na camada de enlace por seu endereço MAC de 48 *bits*. Em uma comunicação, quando a camada de rede recebe um pacote das camadas superiores a fim de enviá-lo a um determinado endereço IP, ela verifica se este endereço está na mesma rede local do emissor. Se estiver, o pacote deverá ser entregue a camada de enlace que o envia a interface física apropriada. Para que isto ocorra é necessário que o emissor também já conheça o endereço MAC do destinatário.

Este gerenciamento é feito pelo protocolo de resolução de endereços (ARP), que automaticamente mapeia endereços IP para endereços MAC. Cada *host* na rede possui este mapeamento dentro de uma tabela temporária chamada *cache* ARP. Quando necessário qualquer *host* pode procurar em sua própria tabela ARP para encontrar mapeamentos de endereços. Na Tabela 3 ilustra-se uma tabela *cache* ARP em um *host*.

**Tabela 3** – Exemplo de uma tabela ARP.

Endereço IP	Endereço Mac	Timeout (seg)
10.0.0.15	AB:CE:7E:43:D2:33	120
10.0.0.23	76:A3:E2:7C:D7:58	120
10.0.0.49	F1:E7:D8:19:4C:06	120
10.0.0.17	AB:CE:7E:43:D2:33	120

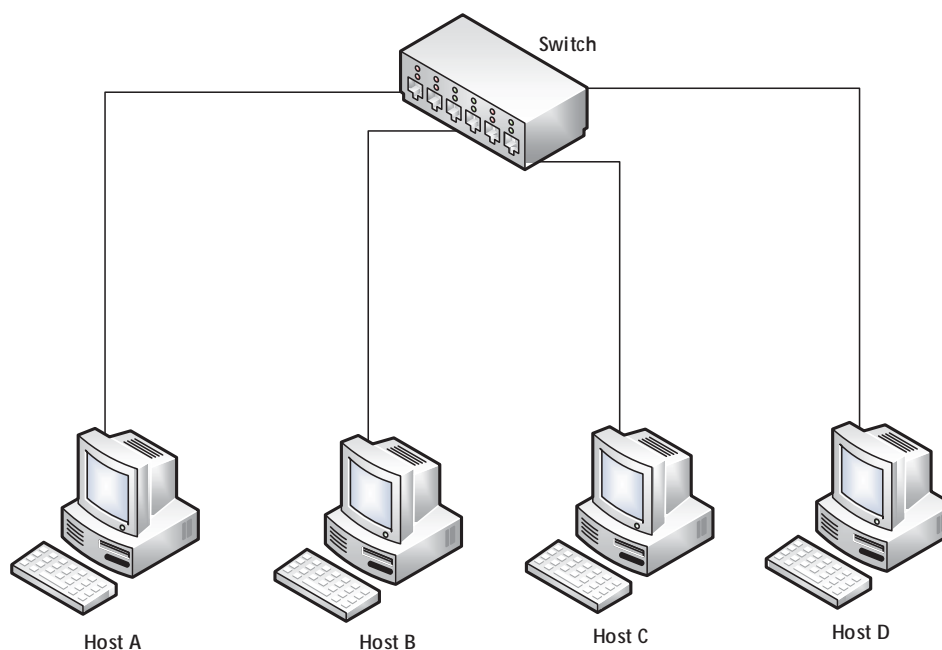
Fonte: Elaboração do autor.



Para gerenciar, adicionando ou excluindo estes mapeamentos na tabela ARP existem dois tipos de mensagens: requisição ARP e resposta ARP.

Quando necessário, se o mapeamento desejado de um endereço MAC para um endereço IP não for encontrado na tabela ARP, uma mensagem de requisição ARP é enviada a todos os *hosts* da mesma rede local do emissor (PHILIP, 2007). Esta mensagem de requisição contém os endereços IP e MAC do emissor solicitante, o tipo de mensagem (ARP *Request*), o endereço IP alvo que se deseja descobrir e o endereço MAC de *broadcast* FF:FF:FF:FF:FF:FF como destino. Ao chegar a cada *host* da rede, a mensagem é analisada por cada um. Se seu próprio endereço IP não for igual ao endereço IP alvo da mensagem recebida, este simplesmente descarta a mensagem, mas se for igual, ele envia ao *host* que emitiu a requisição, uma mensagem de resposta (ARP *Reply*) a fim de informar seu endereço MAC. Quando a mensagem de resposta chegar ao emissor, sua tabela ARP será atualizada com o novo mapeamento. Este procedimento é repetido sempre que necessário para atualizar o *cache* ARP dos vários *hosts* comunicantes. Um exemplo de requisição e resposta ARP é demonstrado a seguir e ilustrado na Figura 23:

1. O *host* A deseja enviar pacotes de dados para o *host* D, mas apenas conhece seu endereço IP e não seu endereço MAC;
2. O *host* A envia uma mensagem de requisição ARP em *broadcast* a todos os *hosts* da rede, questionando qual deles é o *host* com o endereço IP alvo.
3. Todos os *hosts* da mesma rede local recebem a mensagem de requisição e a analisam. Com exceção do *host* D, todos descartam a mensagem de requisição ARP por não conterem o endereço IP alvo;
4. O *host* D responde diretamente ao *host* A uma mensagem de resposta ARP contendo seu endereço MAC e também atualiza sua própria tabela ARP criando um mapeamento com os endereços do *host* A;
5. O *host* A recebe a resposta ARP e atualiza sua tabela ARP com os endereços MAC e IP do *host* D;
6. A partir deste momento os *hosts* A e D podem entregar pacotes diretamente um ao outro, pois suas respectivas tabelas ARP contém os mapeamentos de endereços adequados.

**Figura 23** – Topologia de uma rede local.

Fonte: Elaboração do autor.

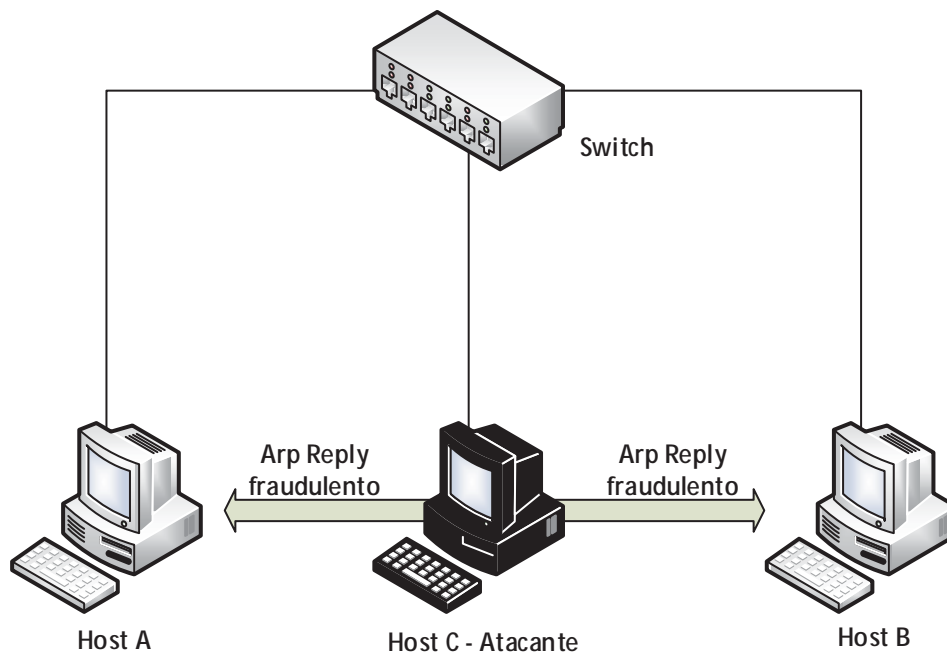
Uma importante observação é que estas entradas de mapeamento dentro das tabelas ARP não são permanentes, assim após algum tempo, aproximadamente dois minutos, os mapeamentos vencidos serão excluídos e os *hosts* deverão repetir o procedimento mencionado para atualizar seus caches ARP novamente.

### 3.2 ATAQUES DE ENVENENAMENTO DE *CACHE* ARP E O ATAQUE MITM

O envenenamento de *cache* ARP acontece quando um atacante maliciosamente modifica o mapeamento de um endereço IP para seu próprio endereço MAC na tabela ARP de outros *hosts* (PHILIP, 2007). Esta técnica, também chamada de fraude ARP (*ARP spoofing*), ocorre com o envio de respostas ARP para todos os *hosts* em uma rede local de forma indiscriminada e mesmo sem requisições anteriores.

Como mostrado na Figura 24, o *host C* onde está o atacante, envia mensagens de resposta ARP para os *hosts A* e *B*, mapeando seu endereço MAC para os endereços IP de cada uma das vítimas. A mensagem enviada para *B* contém o endereço IP de *A* e o endereço MAC de *C* como mapeamento e a mensagem enviada para *A*, contém o endereço IP de *B* e novamente o endereço MAC de *C*.

**Figura 24** – *Host* atacante em uma rede local.



Fonte: Elaboração do autor.

Como resultado A e B atualizarão suas respectivas tabelas ARP com os mapeamentos fraudulentos. Assim quando A enviar pacotes de dados para B, estes conterão o endereço MAC de destino do atacante e o mesmo acontece quando B for enviar pacotes ou responder para A. Com isso o atacante se instala no meio da comunicação dos *hosts* A e B e todo tráfego recebido por ele de alguma das vítimas poderá ser copiado e repassado ao destino de forma discreta. Como o atacante repassa os pacotes de dados após ter lido ou copiado, A e B conseguem se comunicar normalmente e, portanto, podem nem perceber que são vítimas do conhecido ataque MITM.

### 3.3 REDES DEFINIDAS POR SOFTWARE E O ATAQUE MITM

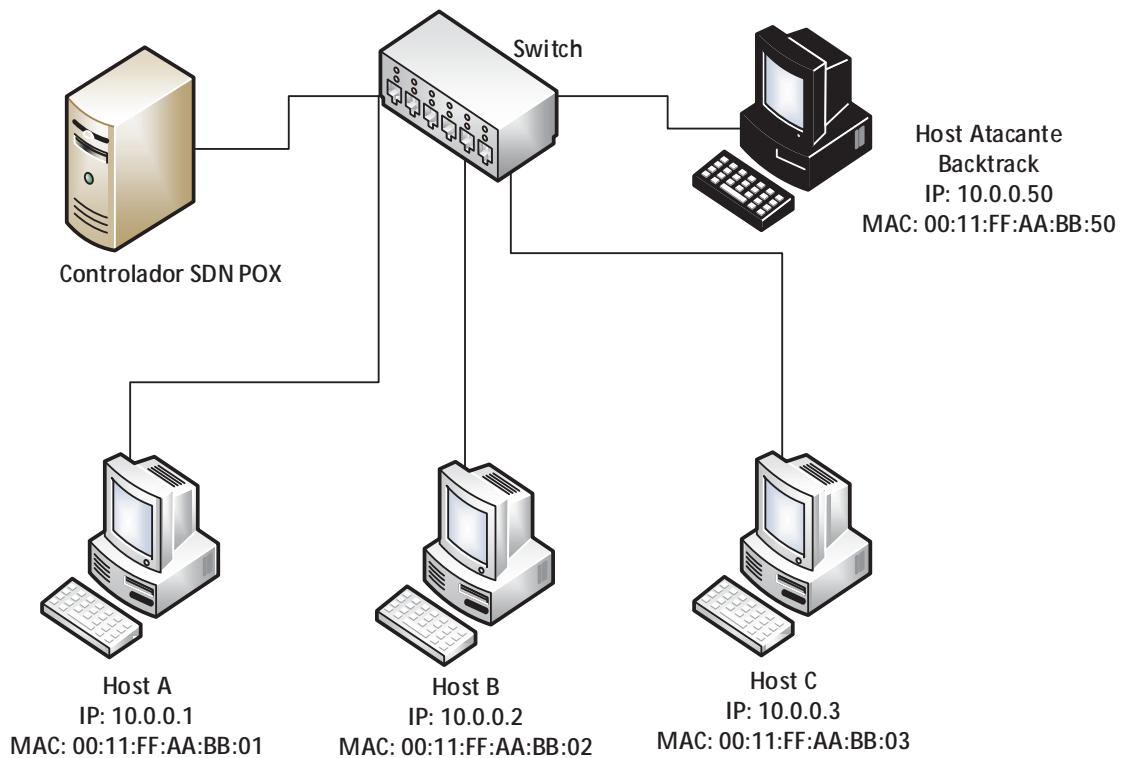
SDNs possuem como funcionalidade mais importante, a possibilidade de programação de regras e políticas no repasse e roteamento dos pacotes. Essas regras são implantadas no controlador e este gerencia os *switches*. Explorando estas características deste novo paradigma, mais adiante neste trabalho será proposto um módulo controlador utilizando o protocolo de redes definidas por software *OpenFlow* (MCKEOWN et al., 2008), a fim de mitigar o ataque MITM em redes locais.

### 3.3.1 Cenário de testes

Para a realização dos testes de ataque MITM de forma prática neste novo paradigma, foi criada uma rede virtual no simulador SDN Mininet (LANTZ; HELLER; MCKEOWN, 2010; OLIVEIRA et al., 2014) contendo: 3 *hosts*, 1 *switch*, 1 *host* atacante rodando o sistema operacional Linux com a distribuição BackTrack5 e um controlador POX. Na Figura 25 ilustra-se o esquema mencionado.

O BackTrack5 é uma distribuição do sistema operacional Linux específica para realização de testes e estudos relacionados à segurança. Ao instalá-la, um vasto conjunto de ferramentas de rede, criptografia, auditoria e de testes fica disponível.

**Figura 25** – Cenário para realização de testes.



Fonte: Elaboração do autor.

No primeiro teste foi configurado no controlador POX, o módulo `L2_learning`, cuja programação implementa características semelhantes à de um *switch* de segunda camada comum em redes locais. O Apêndice A, contém o código-fonte completo do módulo `L2_learning`.

Com a rede em funcionamento, foram executados testes de conexão através de mensagens ICMP *ECHO Request (ping)* entre todos os *hosts*, sendo confirmada a comunicação entre eles. Após alguns instantes analisando pacotes entrantes nas diversas portas do *switch*, o controlador configurou regras de fluxo no *switch* de modo que os *hosts* pudessem trocar informações diretamente e sem que os pacotes fossem repassados para todas as portas do *switch* todas as vezes. Com essas regras no *switch*, os pacotes trocados entre os *hosts* A e B passam somente pelas portas 1 e 2 do *switch*, portanto nem o *host* C nem o *host* atacante conseguem visualizar tais pacotes. Este recurso de auto aprendizado é muito importante, pois as regras implementadas no *switch* fazem com que os próximos pacotes que forem trafegar entre *hosts* já conhecidos não precisem ser pesquisados no controlador, o que garante o desempenho no repasse dos pacotes na rede.

Posteriormente com a rede ainda em funcionamento, foi desferido o ataque MITM do *host* atacante contra os *hosts* A e B, através da execução dos seguintes comandos:

```
# arpspoof -i eth0 -t 10.0.0.1 10.0.0.2  
# arpspoof -i eth0 -t 10.0.0.2 10.0.0.1
```

O parâmetro `-i` representa a interface local utilizada para desferir o ataque, neste caso `eth0`. O parâmetro `-t` representa o endereço IP que se deseja fraudar, neste caso tanto `10.0.0.1` quanto `10.0.0.2`. De uma maneira simples os comandos podem ser expressados como: “Diga para o endereço IP `10.0.0.1` que eu (atacante) sou o endereço IP `10.0.0.2` e diga para o endereço IP `10.0.0.2` que eu sou o endereço IP `10.0.0.1`”.

Estes comandos enviam mensagens de respostas ARP para os *hosts* A e B de forma contínua e a cada um segundo, o que faz com que as tabelas ARP de ambos sejam atualizadas com o endereço MAC fraudulento do atacante. Após isto, toda comunicação entre as vítimas é interceptada pelo atacante, que rapidamente repassa todos os pacotes para o destino correto a fim de não ser notado.

### 3.3.2 Conclusão

Nota-se nesta seção que ambas as estruturas de redes locais, a tradicional e a definida por software, estão sujeitas ao ataque MITM. O protocolo ARP foi implementado nesses primeiros testes em redes SDNs sem mudança alguma e isto o deixou igualmente vulnerável.

Uma grande vantagem das redes definidas por software em relação ao padrão atual é exatamente a possibilidade de programar funcionalidades no componente controlador. A próxima seção apresenta uma proposta para mitigar o ataque MITM e ainda controlar as mensagens ARP na rede utilizando esta possibilidade.

## 4 O MÓDULO L3-ARPSEC

O módulo L3-ARPsec é um conjunto de instruções escritas em linguagem de programação Python que é executado no controlador. Ele propõe uma maneira de controlar a troca de mensagens ARP e ao mesmo tempo combater o ataque MITM em redes locais. Desenvolvido exclusivamente pelo autor deste trabalho, o código-fonte completo do módulo, com todas as suas funcionalidades se encontra no Apêndice B deste trabalho.

As premissas a seguir são fundamentos deste módulo e norteiam o seu funcionamento:

- 1 Não são necessárias alterações no protocolo ARP original, desta forma, nenhum *host* precisa ser configurado individualmente;
- 2 Todo pacote do tipo ARP que chegar a qualquer uma das portas do *switch* será encaminhado para tratamento no controlador;
- 3 Somente o controlador pode enviar mensagens do tipo resposta (ARP *Reply*), portanto nenhum *host* da rede responderá a um ARP *Request* de outro *host* diretamente;
- 4 Reconhecer automaticamente os endereços dos *hosts* e enviar ao *switch* regras de fluxo para melhorar o desempenho no repasse dos pacotes da rede.

### 4.1 TABELAS VIRTUAIS DE MAPEAMENTO DE ENDEREÇOS NO CONTROLADOR

Ao ser executado no controlador, o módulo L3-ARPsec, instancia duas tabelas virtuais ARP: uma chamada *ArpTable* e outra chamada *ArpTableCandidate*. As estruturas destas tabelas estão ilustradas na Tabela 4 e como elas são utilizadas será explicado mais adiante.

Uma característica padrão do protocolo *OpenFlow* é que quando um pacote chega a qualquer uma das portas do *switch* e não encontra nenhuma regra de fluxo coincidente, ele envia este pacote para tratamento no controlador. No fluxograma da Figura 26 pode ser visualizado como o módulo L3-ARPsec trata cada pacote recebido e a seguir os detalhes do módulo são explicados.

**Tabela 4** – Estrutura das tabelas virtuais.

<b>Endereço IP</b>	<b>Endereço MAC</b>	<b>Porta no Switch</b>	<b>Timeout (seg)</b>
10.0.0.15	AB:CE:7E:43:D2:33	12	60
10.0.0.23	76:A3:E2:7C:D7:58	8	60
10.0.0.49	F1:E7:D8:19:4C:06	20	60
10.0.0.17	AB:CE:7E:43:D2:33	10	60

Fonte: Elaboração do autor.

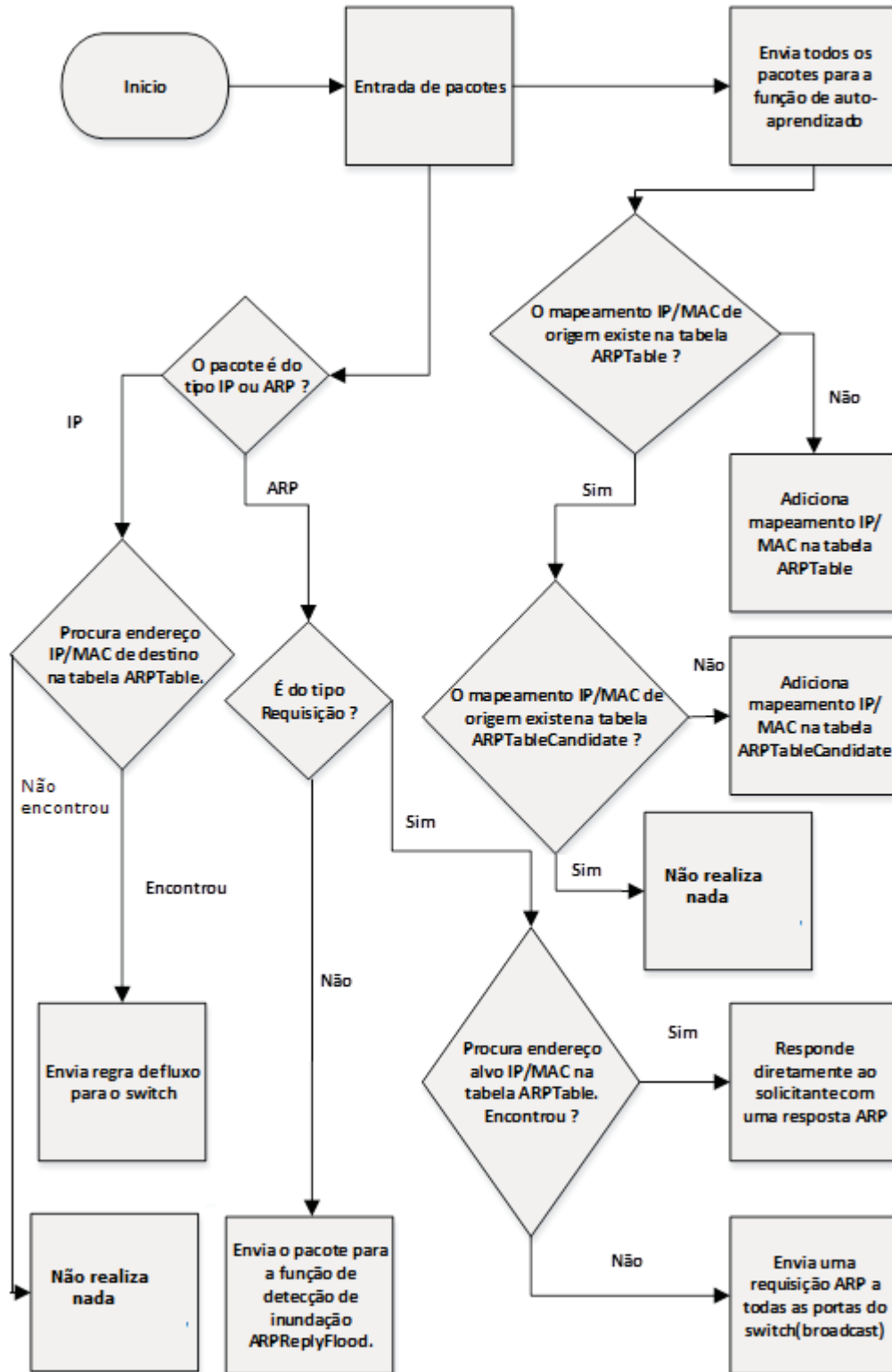
#### 4.2 ENVIO DE TODO PACOTE PARA A FUNÇÃO DE AUTO-APRENDIZADO

Independentemente do tipo de pacote recebido e de sua origem, todo pacote recém-chegado ao controlador será primeiramente encaminhado a função de auto-aprendizado.

Esta função extrai do cabeçalho do pacote os endereços IP e MAC de origem e o número da porta do *switch* pela qual este pacote entrou. Com estes dados procura-se pelo endereço IP na tabela ARPTTable. Caso não seja encontrado, insere um novo mapeamento nesta tabela. Caso seja encontrado, não insere na tabela ARPTTable e procura-se na tabela ARPTTableCandidate, inserindo um novo mapeamento nesta, somente se não encontrado. Como pode ser percebido, esta função é responsável por alimentar os mapeamentos nas tabelas ARPTTable e ARPTTableCandidate. Estes mapeamentos serão utilizados posteriormente por outras funções para controlar o fluxo de mensagens ARP na rede.



Figura 26 - Fluxograma do módulo L3-ARPSec.



Fonte: Elaboração do autor.

### 4.3 VERIFICAÇÃO SE O PACOTE É DO TIPO IP OU ARP

Os mesmos pacotes enviados anteriormente para a função de auto aprendizado também serão enviados para uma função que inicialmente verifica pelo cabeçalho se o pacote é do tipo ARP ou IP.

Caso seja do tipo IP procura-se na *ArpTable* por uma entrada correspondente ao endereço IP. Se for encontrado, cria uma mensagem de fluxo de repasse baseada nos dados de origem do pacote recebido mais os dados de destino encontrados na entrada da tabela *ARPTTable* e envia ao *switch*. O resultado é que os próximos pacotes vindos da mesma origem e com o mesmo destino não precisarão mais passar pelo controlador, eles serão repassados automaticamente pelo *switch*. Isto representa um ganho de desempenho. Esta regra de fluxo instalada no *switch* possui um tempo de inutilização chamado *idle\_time* e caso esta regra fique sem ser usada por mais de vinte segundos ela será descartada.

No caso do pacote recebido ser do tipo ARP, verifica-se ainda se ele é do tipo requisição ou resposta ARP. Caso seja uma mensagem de requisição, procura-se pelo endereço IP alvo na tabela *ARPTTable*, se for encontrado cria um pacote de resposta ARP e envia-o diretamente para o *host* que a requisitou. Se não for encontrado na tabela *ARPTTable*, cria um pacote de requisição ARP e envia em *broadcast* para todas as portas do *switch*. Desta maneira todos os *hosts* irão receber, tratar e no caso do *host* correspondente, responder a requisição.

Se o pacote representar uma resposta ARP, apenas envia o pacote para a função de detecção de inundação de *ARPReply*, esta função contabiliza o recebimento deste tipo de pacote calculando o intervalo médio de tempo entre a chegada deles. A cada cinco pacotes contabilizados, divide-se o intervalo de tempo entre o primeiro e o quinto pacote recebido por cinco, conforme equação (1) a seguir.

$$i = \frac{\Delta t}{5} \quad (1)$$

Sendo que  $i$  representa o valor do intervalo médio de tempo e  $t$  representa o valor do tempo entre a chegada do primeiro e o quinto pacote.

Se este resultado for menor que três segundos, significa que algum *host* está inundando a rede com pacotes de respostas ARP e isto se caracteriza um ataque. A escolha deste intervalo médio de tempo se fundamenta, pois, em situações normais de funcionamento, requisições ou respostas ARP não são enviadas pelos *hosts* em grande escala e em pequenos espaços de tempo. Estes valores representam uma constante inicial para o algoritmo e podem ser ajustados em pesquisas futuras.

Caracterizado o ataque pela função explicada, é invocada uma rotina que bloqueia temporariamente este endereço MAC emitente. O controlador envia ao *switch* uma regra de fluxo que permanecerá por dois minutos apagando todos os pacotes que chegarem a qualquer uma das portas e possuam como endereço MAC de origem o mesmo do ataque detectado. Além disso, qualquer entrada nas tabelas que contenham este endereço MAC será excluída.

**Tabela 5** – Tabela contendo a quantidade de respostas ARP por endereço IP.

Endereço IP	Endereço Mac	Qtde	Primeiro registro
10.0.0.1	00:11:FF:AA:BB:50	5	08:01:00
10.0.0.2	00:11:FF:AA:BB:02	3	08:03:05
10.0.0.3	00:11:FF:AA:BB:03	1	08:01:30
10.0.0.50	00:11:FF:AA:BB:50	2	08:02:22

Fonte: Elaboração do autor.

Por exemplo, se um determinado *host* envia na rede respostas ARP a cada três segundos e de maneira contínua, o tempo entre o primeiro e o quinto pacote recebido será de doze segundos. Ao dividir doze por cinco, se obtém o valor de 2,4 e isto significa que provavelmente este *host* está deflagrando um ataque de inundação ARP. Em condições normais de tráfego na rede, as respostas ARP enviadas pelos *hosts* são eventuais e em pequenas quantidades, desta forma esta função não bloqueará o tráfego normal de mensagens ARP.

A tabela utilizada para a contabilização dos pacotes a fim de detectar o ataque de inundação de respostas ARP é ilustrada na Tabela 5.

#### 4.4 TIMERS DE VERIFICAÇÃO

Em um ambiente de rede local é possível que os *hosts* troquem de endereço IP, de porta de conexão no *switch* ou até mesmo de endereço MAC, caso alguma interface física seja trocada. Se as entradas nas tabelas ARPTTable e ARPTTableCandidate fossem permanentes, esta troca não seria possível e sempre o primeiro registro para um *host* não permitiria alterações futuras. Para que isso não ocorra existe em cada entrada nas tabelas além do número da porta no *switch* e dos endereços IP e MAC, um valor de *timeout*, que corresponde ao tempo de vida em segundos que esta entrada terá. Uma rotina executada continuamente a cada vinte e cinco segundos verifica cada entrada nas duas tabelas e exclui aquelas cujo *timeout* tenha se esgotado.

Também é possível que em alguns casos, a função de detecção de inundação de ARPReply explicada anteriormente, não consiga detectar o ataque, pois o atacante pode emitir mensagens de requisição ARP em frações de segundos maiores do que a média tolerável pelo algoritmo de detecção da função. Pensando nisto, outra forma de detecção dos ataques ARP é avaliar as entradas nas tabelas ARPTTable e ARPTTableCandidate.

Uma rotina executada a cada treze segundos faz uma varredura nas tabelas procurando por entradas com endereços IP diferentes, mas com o mesmo endereço MAC. Se forem encontradas significa que algum *host* está utilizando seu endereço próprio IP e também está tentando se passar por algum outro *host* da rede. Isto caracteriza um ataque de falsificação ARP e a mesma rotina de bloqueio utilizada pela função de detecção de inundação ARPReply será invocada, bloqueando temporariamente o endereço MAC que desferiu o ataque.

#### 4.5 EXECUÇÃO DE TESTES COM O MÓDULO L3-ARPSEC

Utilizando o mesmo cenário da subseção 3.3.1, contendo três *hosts*, um *switch*, um controlador e um *host* atacante, alguns testes foram realizados para verificar o funcionamento na prática do módulo proposto.

Primeiramente, o módulo foi configurado e devidamente iniciado no controlador da rede. Com isto os *hosts* puderam trocar pacotes entre si de forma transparente, provando que o controlador conseguiu reconhecer os *hosts* e enviar ao *switch* regras de fluxo para o encaminhamento dos pacotes.

#### 4.5.1 Testando a função de detecção de inundação ARPReplayFlood

O primeiro teste de segurança com o módulo foi verificar a sua capacidade de detecção do ataque ARP *flooding*, na qual um dos *hosts* da rede, neste caso o atacante utilizando a ferramenta *arpspoof* do sistema operacional Linux Backtrack, inunda a rede enviando mensagens de resposta ARP de forma constante e a cada um segundo. A linha de comando executada no atacante pode ser visualizada a seguir.

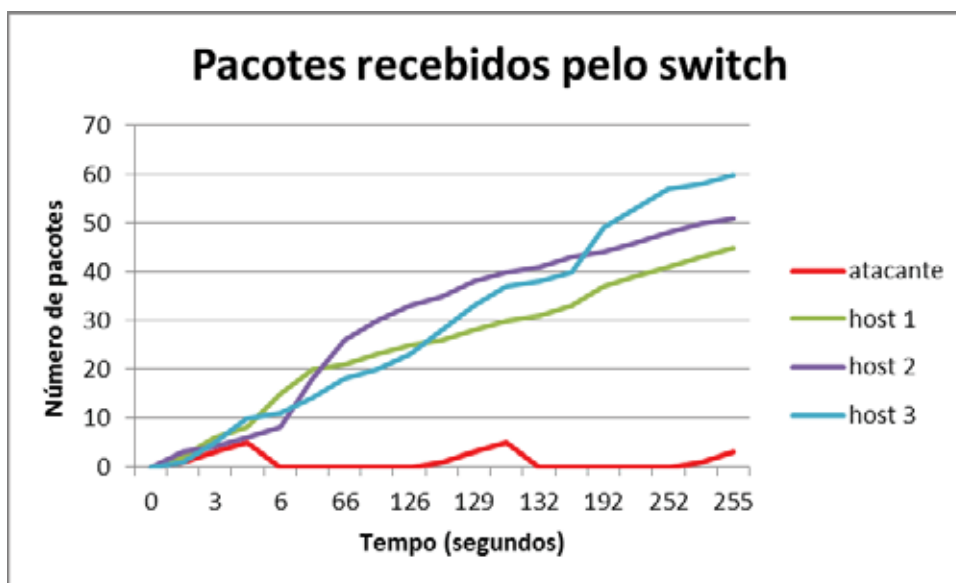
```
# arpspoof -i eth0 -t 10.0.0.1 10.0.0.2
```

Cada uma das mensagens de resposta ARP chegou ao *switch* e como não havia nenhuma regra por padrão que pudesse ser seguida, ele repassou os pacotes ARP para o controlador. O módulo L3-ARPSec então configurado no controlador, através da função de detecção de inundação ARPReplayFlood, foi contabilizando cada mensagem recebida e quando a quinta mensagem idêntica foi processada, o ataque foi detectado.

A reação frente à detecção do ataque foi o envio de uma regra de fluxo ao *switch* a fim de bloquear temporariamente o atacante emissor das mensagens fraudulentas. Ao receber esta regra, o *switch* passou instantaneamente a apagar todo e qualquer pacote vindo daquele emissor, deixando-o totalmente inerte na rede por um período de dois minutos. Os mapeamentos existentes nas tabelas virtuais ARPTable e ARPTableCandidate contendo o endereço MAC do atacante também foram excluídos para que nenhum outro *host* fosse enganado por este endereço fraudulento.

Utilizando o aplicativo *wireshark*, uma das ferramentas já explicadas em seções anteriores, foram coletados dados de tráfego na rede durante aproximadamente cinco minutos (300 segundos). Os dados foram compilados em um gráfico de linha e mostrados na Figura 27.

Figura 27 – Dados coletados durante tentativa de ataque.



Fonte: Elaboração do autor.

#### 4.5.2 Resultados e discussão do primeiro teste

É possível notar que no decorrer do tempo, os *hosts* 1, 2 e 3 foram aumentando gradativamente o fluxo de pacotes enviados ao *switch*. No entanto, o *host* atacante durante os aproximados cinco minutos de monitoramento da rede, foi bloqueado várias vezes. Como a função de detecção de inundação ARPReplyFlood contabiliza de cinco em cinco pacotes e o atacante estava enviando um pacote a cada segundo, então o bloqueio se dava no decorrer de cinco segundos. É perceptível no gráfico também que nos próximos dois minutos após o bloqueio, o atacante não conseguiu enviar nenhum pacote ao *switch*, mas esgotado este tempo de punição, o ataque voltou a ocorrer no segundo 126. Novamente a função de detecção de inundação ARPReplyFlood começou a contabilizar as respostas ARP enviadas pelo atacante e imediatamente após cinco pacotes recebidos, o bloqueio foi feito, deixando-o punido e inerte na rede por mais dois minutos.

Este ciclo de detecção e bloqueio é feito constantemente, pois o módulo L3-ARPSec está sempre monitorando a rede toda. Em caso de novos atacantes serem adicionados a rede e desferirem ataques deste tipo, eles também serão penalizados com o bloqueio de dois minutos.

É importante também ressaltar que o valor cinco, definido para a contabilização dos pacotes e realização da média de intervalo para a decisão de bloqueio ou não, representa apenas uma escolha inicial ótima, tendo em vista que durante a realização dos testes não houve bloqueio indevido de *hosts* trafegando mensagens ARP não fraudulentas, mas foi

possível bloquear todos os ataques de inundação deflagrados. O valor definido em dois minutos para punição do atacante, também representa uma escolha inicial e em ambos os casos, futuras pesquisas podem encontrar valores melhores para tal função.

Concluindo este primeiro teste, percebe-se o sucesso na detecção do ataque de inundação, porém é válido lembrar que o atacante pode enviar respostas ARP fraudulentas em intervalos de tempo maiores e não ser detectado por este mecanismo do módulo. Na próxima subseção são apresentados os dados de outro teste realizado para verificar a qualidade de detecção dos ataques por meio dos *timers*, o segundo mecanismo de detecção do módulo.

### 4.5.3 Testando os *Timers* para a detecção dos ataques MITM

Neste segundo teste de segurança com o módulo foi verificada a capacidade de detecção do ataque MITM para os casos em que a função de detecção anterior não obteve sucesso. Utilizando o mesmo cenário já configurado anteriormente, o atacante desferiu o ataque de uma forma diferente. Ao invés de inundar a rede enviando mensagens de resposta ARP constantemente a cada um segundo, ele as enviou em intervalos maiores, aproximadamente quatro segundos.

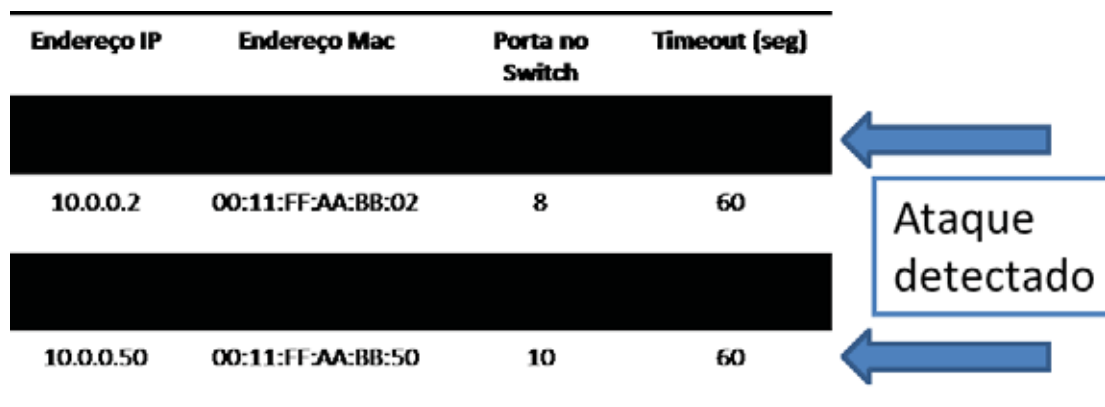
Com as mensagens chegando ao *switch* e sendo repassadas ao controlador neste intervalo de tempo maior, o módulo não reconheceu este tráfego como malicioso e assim os mapeamentos para os endereços fraudulentos foram instalados na tabela virtual ARPTable. O atacante enviou à rede respostas ARP contendo seu próprio endereço IP (10.0.0.50) e endereço MAC (00:11:FF:AA:BB:50), e além disso também enviou respostas ARP fraudulentas contendo o endereço IP da vítima (10.0.0.1) e novamente seu próprio endereço MAC (00:11:FF:AA:BB:50). Na Tabela 6 é possível visualizar as entradas fraudulentas na tabela virtual ARPTable.

Qualquer *host* da rede que quisesse enviar dados ao *host* 1 (10.0.0.1) teria que solicitar seu endereço MAC ao módulo através de mensagens ARP e como o módulo não sabia até o momento da fraude, as respostas às requisições ARP teriam o endereço MAC do atacante (00:11:FF:AA:BB:50).

Como as mensagens fraudulentas foram enviadas pelo atacante em intervalos de quatro segundos aproximadamente, a rotina de verificação das tabelas foi executada no segundo 13 e detectou a fraude. Lembrando que esta rotina é executada no módulo a cada treze segundos e faz uma varredura nas tabelas procurando por mapeamentos contendo endereços IP diferentes, mas com o mesmo endereço MAC.

Tabela 6 – Entradas fraudulentas detectadas na tabela virtual.

Endereço IP	Endereço Mac	Porta no Switch	Timeout (seg)
10.0.0.2	00:11:FF:AA:BB:02	8	60
10.0.0.50	00:11:FF:AA:BB:50	10	60



Fonte: Elaboração do autor.

Ao detectar o ataque, a mesma função de punição utilizada pela detecção de inundação ARPReplyFlood foi acionada, deixando o atacante bloqueado na rede por dois minutos.

#### 4.5.4 Resultados e discussão do segundo teste

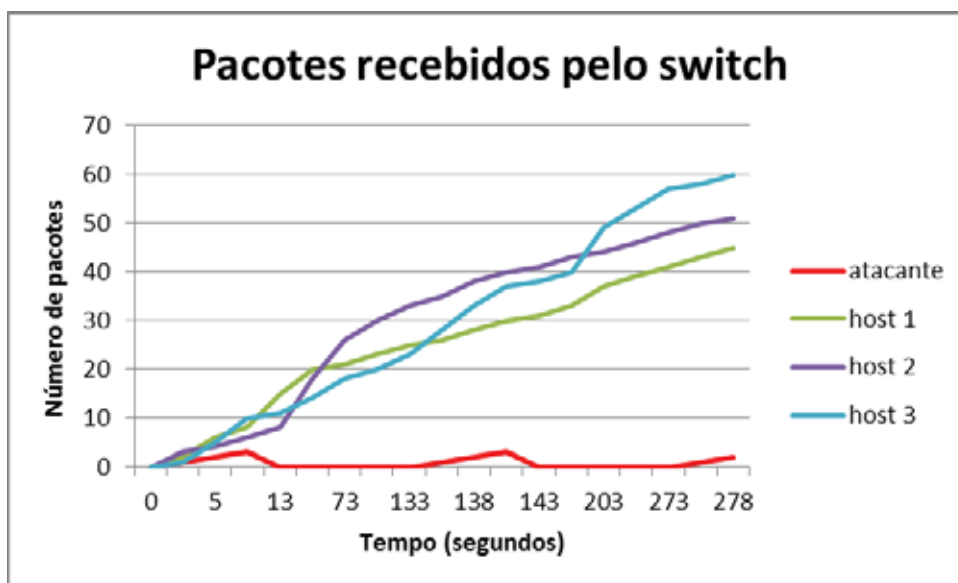
Percebe-se que o atacante apesar de conseguir burlar o primeiro mecanismo de detecção do módulo, o ARPReplyFlood, acabou sendo detectado várias vezes durante os cinco minutos decorridos deste segundo teste. Como a rotina de varredura é executada a cada 13 segundos, o bloqueio se deu logo após a entrada do mapeamento fraudulento na tabela ARPTable.

É evidente no gráfico da Figura 28 que os próximos dois minutos após cada bloqueio, o atacante não conseguiu enviar nenhum pacote ao *switch*, mas esgotado este tempo de punição, a tentativa de ataque voltou a ocorrer no segundo 134. Os mapeamentos fraudulentos foram instalados na tabela ARPTable e a varredura do segundo 143 detectou novamente o ataque, bloqueando e punindo o atacante na rede por mais dois minutos.

Este ciclo de detecção e bloqueio é feito constantemente, pois o módulo L3-ARPSec está continuamente executando esta rotina de varredura a cada treze segundos. Se novos atacantes forem adicionados à rede e desferirem ataques deste tipo, eles também serão penalizados com o bloqueio de dois minutos.



**Figura 28** – Dados coletados durante tentativa de ataque.



Fonte: Elaboração do autor.

É importante também ressaltar que o valor de 13 segundos definido para o intervalo de cada varredura, representa apenas uma escolha inicial ótima, tendo em vista que durante a realização dos testes não houve perda de desempenho do módulo por conta do processamento intenso da varredura e foi possível bloquear todos os ataques deflagrados.

Concluindo este segundo teste, percebe-se o sucesso na detecção dos ataques utilizando os *timers*. As tentativas de fraudes que escaparam da detecção do primeiro mecanismo foram detectadas pelo segundo que se mostrou muito eficiente.

#### 4.5.5 Tempo de bloqueio adaptativo para ataque recorrente.

Tanto a Função de Detecção de Inundação ARPReplayFlood quanto a função de detecção por meio dos *Timers*, ao perceberem a existência do ataque na rede, invocam uma função de punição. Como já explicado, esta função bloqueia totalmente o endereço MAC do atacante na rede pelo prazo inicialmente definido de dois minutos. Após este período, o endereço punido é liberado e novamente poderá trocar dados na rede.

O bloqueio temporário com subsequente liberação e a não punição definitiva, se baseia no fundamento de que um determinado computador pode ser usado por um atacante especificamente em um momento, mas posteriormente este mesmo computador poderá ser utilizado por um outro usuário confiável na mesma rede. Em contrapartida, isto possibilita o

atacante continuar atacando insistentemente e neste caso, ao terminar o período de bloqueio de dois minutos, ele retoma o ataque novamente.

Pensando nesta hipótese, o algoritmo que bloqueia o endereço MAC na rede é adaptativo e incrementa o tempo de punição em caso de reincidência. Para isso, uma tabela virtual é instanciada contendo o endereço MAC punido e a respectiva quantidade de vezes que ele já foi penalizado.

Sempre que for necessário bloquear algum endereço MAC na rede, o algoritmo primeiramente procura nesta tabela pelo endereço em questão. Caso nenhuma correspondência seja encontrada, significa que esta é a primeira vez que ele está sendo punido, portanto além de punir o endereço atacante por dois minutos, adiciona-se nesta tabela o seu endereço MAC e na respectiva quantidade insere-se o número um. Caso seja encontrada uma correspondência, caracteriza-se a reincidência. Portanto, além de incrementar a quantidade de punições para o endereço, o tempo de bloqueio é definido multiplicando-se os dois minutos iniciais pela quantidade de vezes que este já foi bloqueado, incluindo-se esta última vez. Por exemplo: na quarta vez que o atacante for bloqueado, o tempo de suspensão será de oito minutos.

Na Tabela 7 está ilustrada a tabela de punição mencionada.

**Tabela 7** – Tabela de punição adaptativa.

Endereço Mac	Quantidade de punições
AB:CE:7E:43:D2:33	1
76:A3:E2:7C:D7:58	4
F1:E7:D8:19:4C:06	2
CA:5E:87:43:D2:EE	5

Fonte: Elaboração do autor.

#### 4.5.6 Avaliando o desempenho do módulo.

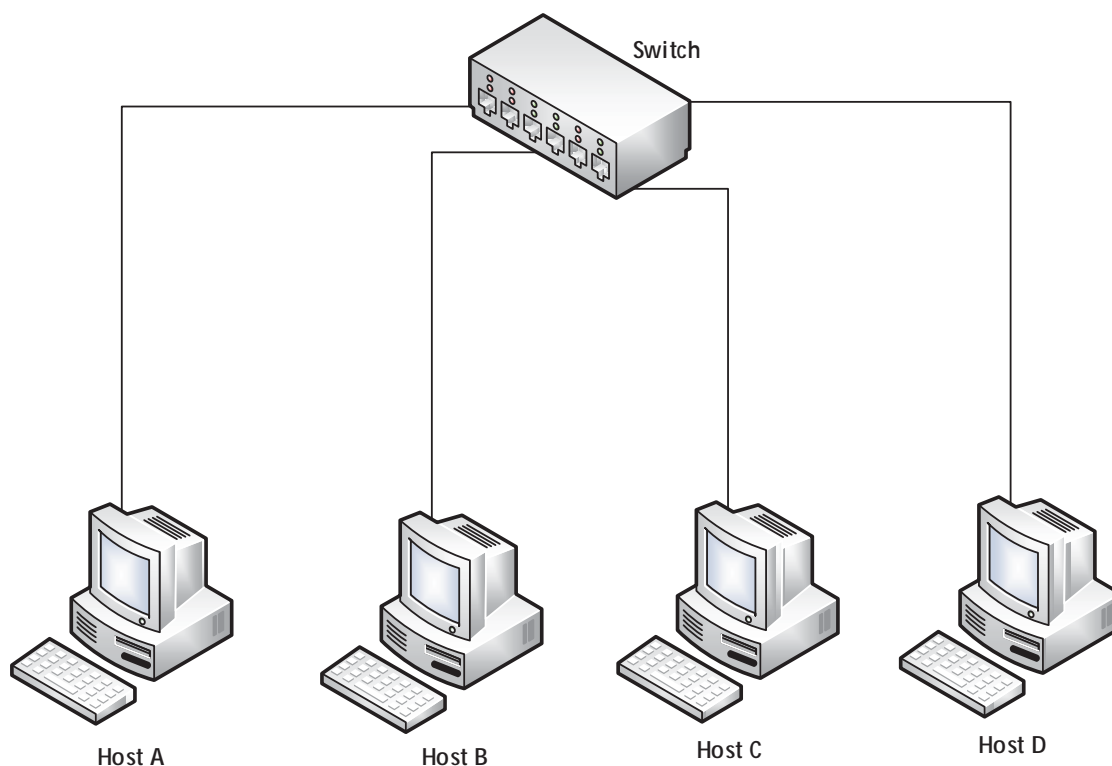
Após realizados testes que verificaram as funcionalidades de segurança do módulo proposto neste trabalho, se fez necessário avaliar seu desempenho e compara-lo a uma rede com estrutura tradicional, pois de nada adianta possuir um mecanismo que garanta a segurança frente a um determinado ataque, mas não garanta desempenho suficiente nas

operações básicas de repasse e roteamento de pacotes na rede. Se o módulo obtiver desempenho negativo frente a demanda da rede, os administradores poderão ficar desencorajados em utilizá-lo.

#### 4.5.6.1 Desempenho utilizando uma rede tradicional.

Para a execução deste teste, foi criada uma rede local semelhante a estrutura de um laboratório de informática no modelo tradicional. Os componentes desta estrutura estão ilustrados na Figura 29 e explicados a seguir.

**Figura 29** – Estrutura tradicional para execução de teste.



Fonte: Elaboração do autor.

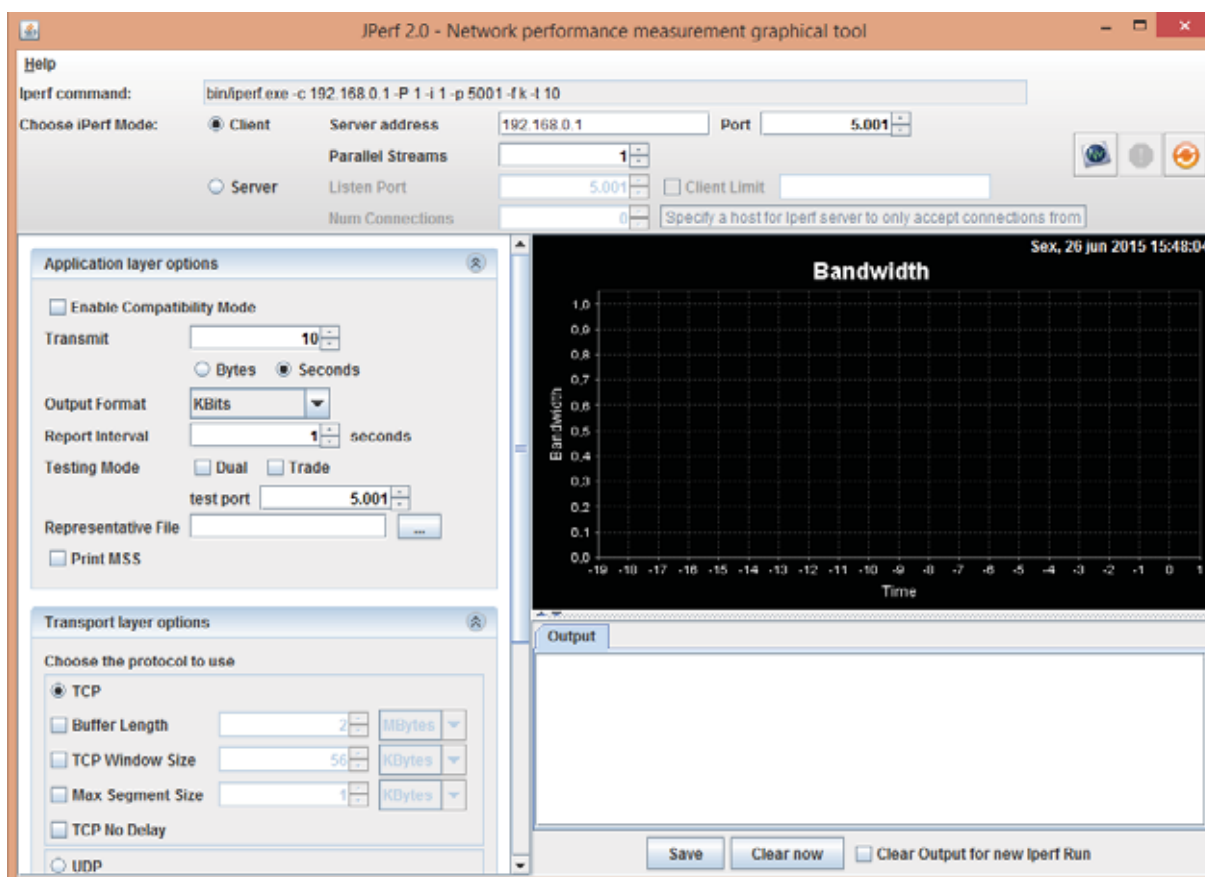
O equipamento adotado para os testes foi o *switch* de marca TP-Link, modelo TL-WR1043ND que contém cinco portas de conexão do tipo ethernet cabeada e ainda permite comunicação de dispositivos sem fio. Inicialmente, nenhuma alteração foi feita no sistema operacional do aparelho, desta forma ele repassa os pacotes de uma porta a outra no modelo tradicional, de forma independente. Todos os microcomputadores possuem como processadores Intel® Core™ i5-2400 CPU @ 3.10GHz, 4 GB de memória RAM, disco rígido

de 300 GB e executam Sistema operacional Windows 7 64 bits. Os cabos permitem a transferência de até cem Megabits por segundo entre todos os nós da rede.

Todos os computadores foram configurados manualmente com endereços IP na faixa de 192.168.0.0/24 e para não influenciar nos resultados dos testes, foram desativados quaisquer outros programas que utilizassem fluxo de rede.

A ferramenta utilizada para medir a velocidade da troca de bits entre os dispositivos foi a IPERF (BLUM, 2003), composta de um aplicativo servidor e um aplicativo cliente, que após configurados os parâmetros básicos, como, a quantidade de dados que se deseja passar de um dispositivo a outro, a porta de comunicação e o tipo de conexão (TCP ou UDP), o aplicativo cliente começa a enviar os bits pela rede para o aplicativo servidor e ambos registram a velocidade destes fluxos. Na figura 30 é possível visualizar a tela de configuração dos parâmetros no aplicativo cliente mencionado.

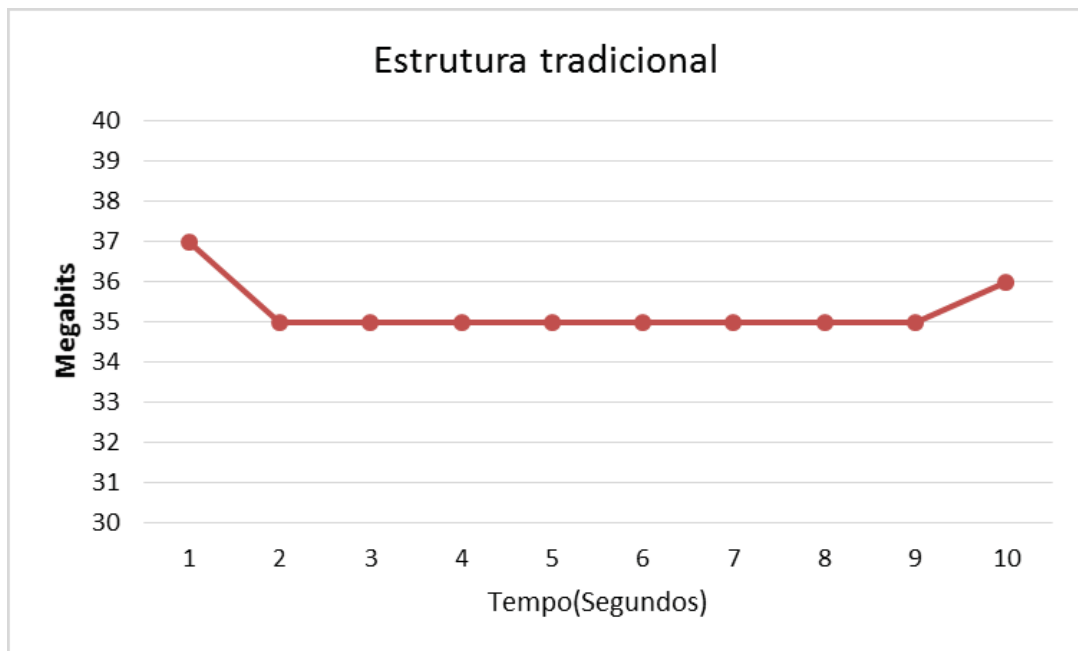
**Figura 30** – Tela de configuração do IPERF.



Fonte: Elaboração do autor.

O aplicativo cliente do IPERF foi instalado e configurado no host A e o aplicativo servidor do IPERF no host B. A porta definida para a comunicação foi a de número 5001, o protocolo de transporte escolhido foi o TCP com um número de dez repetições, uma a cada segundo. Iniciada a transferência dos dados, o aplicativo registrou a velocidade da transmissão dos pacotes que atravessaram o *switch* e o resultado está ilustrado na Figura 31.

**Figura 31** – Velocidade de transmissão na estrutura tradicional.



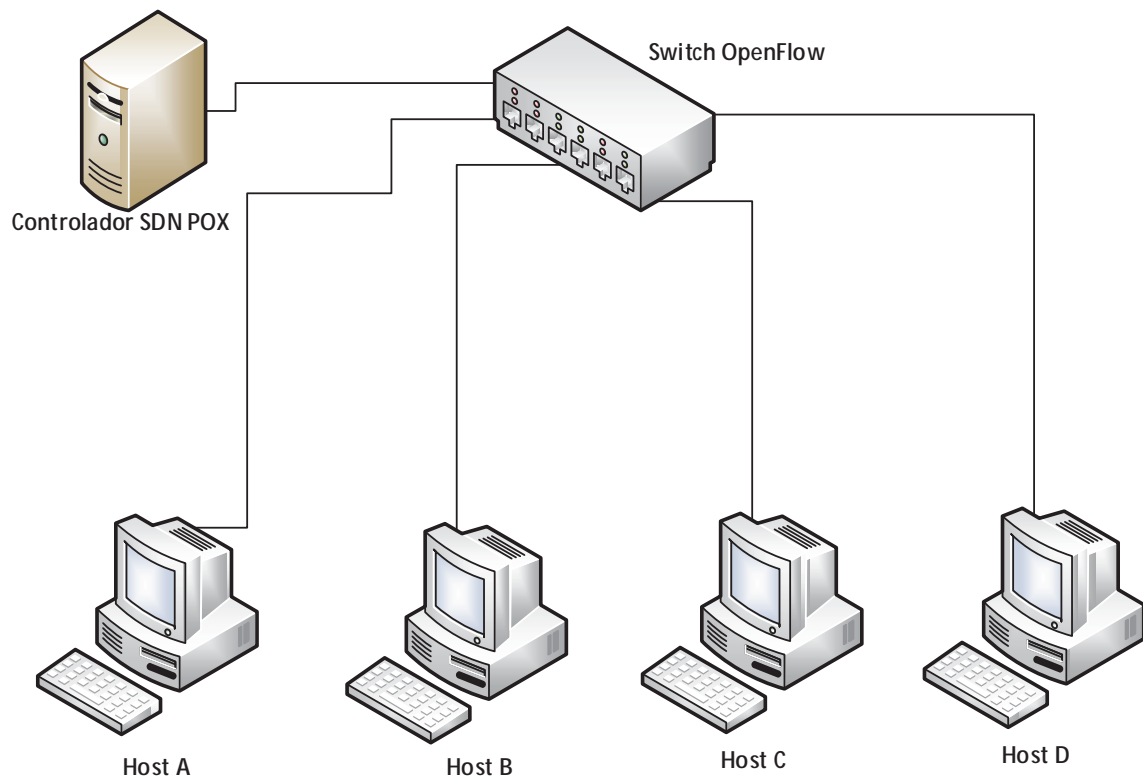
Fonte: Elaboração do autor.

Percebe-se que a transmissão dos dados se deu de maneira uniforme, em torno de 35 megabits por segundo e sem falhas.

#### **4.5.6.2 Desempenho utilizando uma rede overflow e o módulo L3-ARPSec.**

A estrutura para este segundo teste foi configurada de forma similar a utilizada no teste anterior, com exceção de que o *switch* TP-Link teve seu respectivo sistema operacional alterado para suportar o protocolo *OpenFlow* e também foi adicionado a rede, o controlador com o módulo L3-ARPSec implementado. Este controlador foi conectado ao *switch* através de uma porta específica e separada das demais conexões, certificando assim a impossibilidade de ataque diretamente ao mesmo. Uma observação importante é que nesta estrutura, o repasse dos pacotes é gerenciado pelo controlador. A estrutura mencionada pode ser vista na Figura 32.

**Figura 32** – Estrutura SDN para execução de teste.

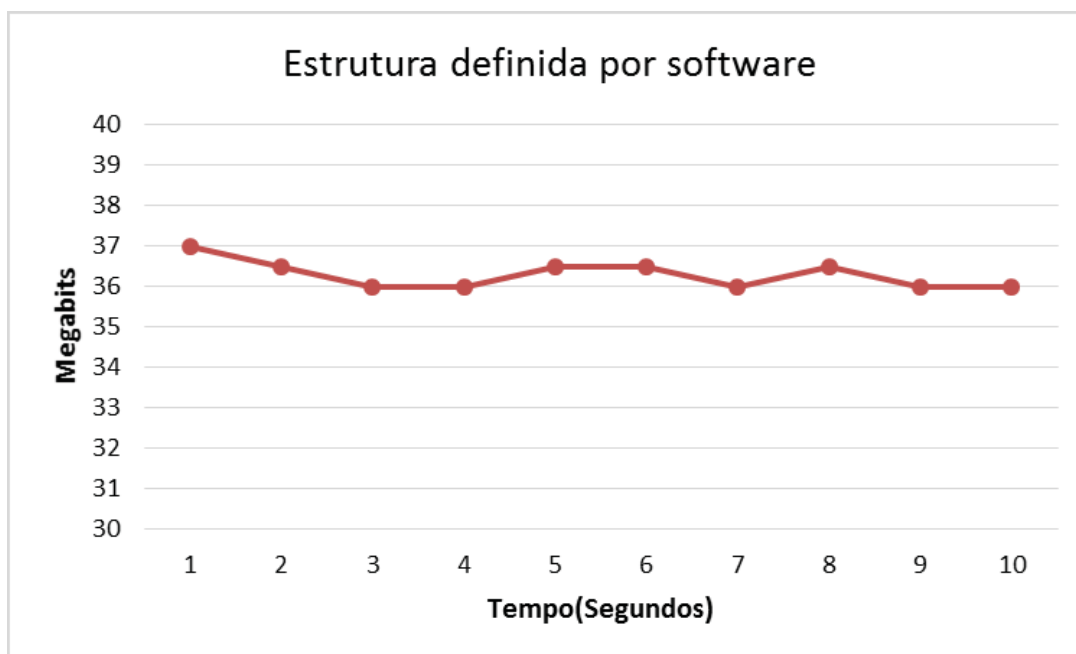


Fonte: Elaboração do autor.

As configurações de porta, endereço IP, protocolo de transporte e demais parâmetros foram mantidos iguais ao teste anterior. A transferência dos pacotes foi registrada e está ilustrada na Figura 33.

Nota-se que a velocidade de transmissão dos pacotes também se manteve uniforme em um número de 36,5 megabits por segundo.

Uma informação relevante é que cada *switch* possui sua própria capacidade de repasse de pacotes, sendo definida pelo poder de processamento do hardware e pela eficiência do software configurado. Como o objetivo foi comparar o desempenho no repasse, nos dois testes realizados o mesmo hardware foi utilizado, ficando a diferença apenas em relação aos softwares implantados.

**Figura 33** – Velocidade de transmissão na estrutura SDN.

Fonte: Elaboração do autor.

#### 4.5.6.3 Desempenho e funcionalidade com vários usuários.

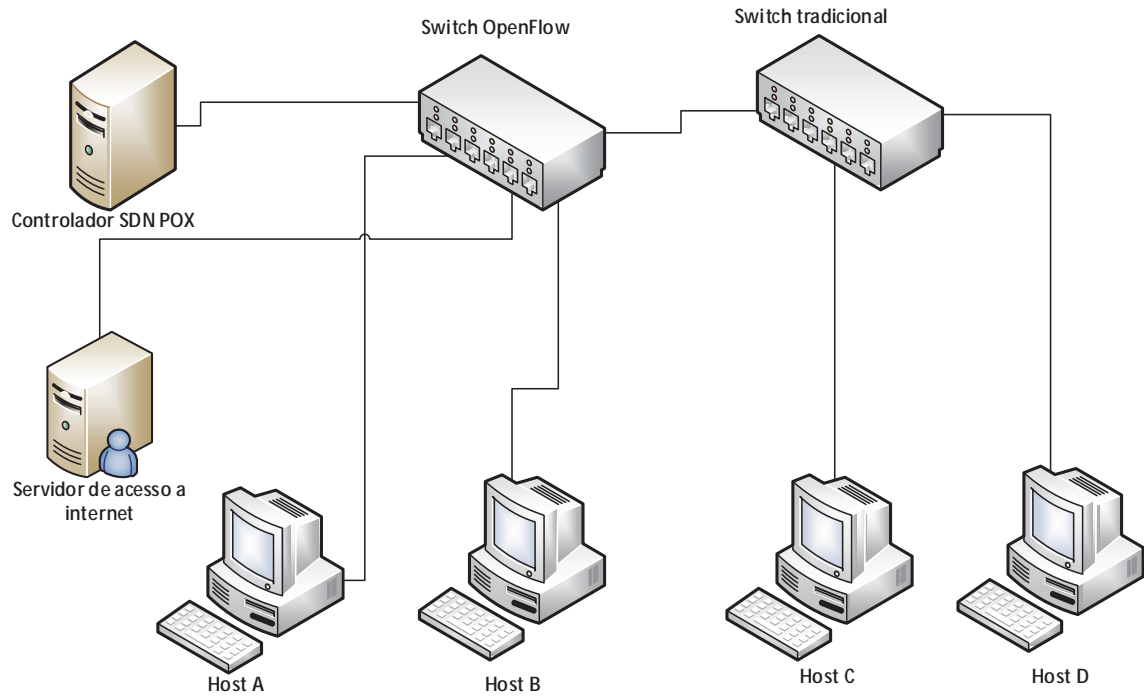
Neste último teste, o objetivo foi ativar e testar o módulo proposto neste trabalho em um laboratório de informática com um número mínimo de vinte usuários simultâneos. Como mostrado na Figura 34, vinte computadores foram conectados a um *switch* tradicional e configurados para acessar a internet exclusivamente através do servidor de acesso. Este servidor de acesso à internet foi conectado a um *switch OpenFlow*, que por sua vez estava conectado a outros dois computadores e também ao *switch* tradicional.

O controlador de rede implementado com o módulo L3-ARPSec foi conectado ao mesmo *switch OpenFlow* a fim de gerenciar o tráfego na rede e bloquear possíveis tentativas de ataques ao protocolo ARP.

Um fato relevante é que nenhum dos usuários foi avisado do teste, desta forma eles não seriam influenciados psicologicamente na coleta dos resultados.

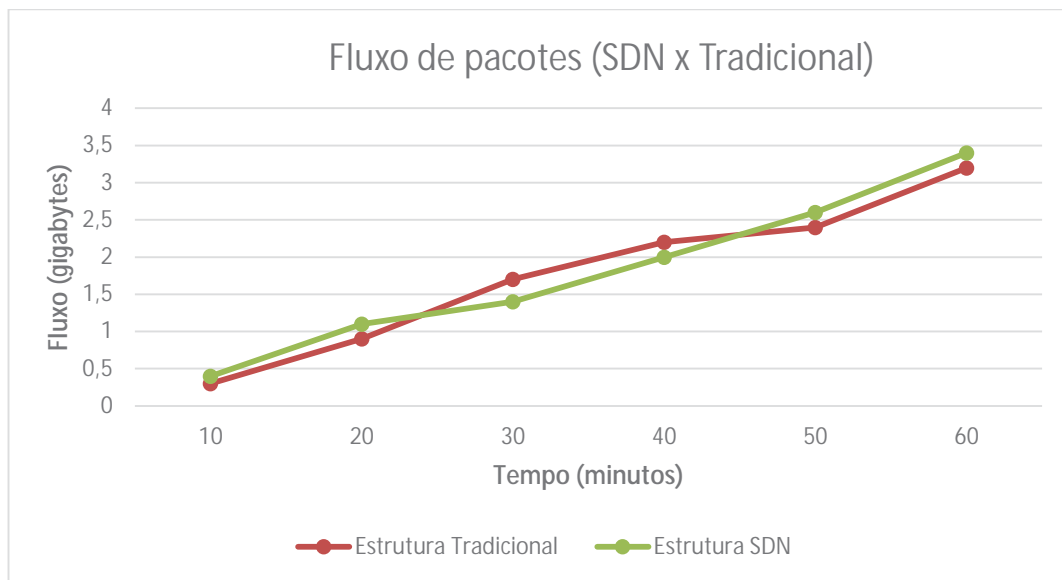
O teste foi realizado pelo período de uma hora ininterrupta e durante todo o tempo a rede foi monitorada. Ao final, os usuários foram questionados sobre a qualidade na utilização do acesso à internet e também sobre possíveis problemas de conexão de rede ocorridos no período. Não foi registrada nenhuma queixa e todos relataram que o acesso ao conteúdo da internet ocorreu de forma tranquila e comum, como em dias anteriores.

**Figura 34 – Estrutura tradicional e SDN.**



Fonte: Elaboração do autor.

**Figura 35 – Fluxo de pacotes registrado no teste.**



Fonte: Elaboração do autor.



O monitoramento da rede também revelou que nenhuma alteração significativa de fluxo ou perda de pacotes ocorreu no período do teste. É possível visualizar na Figura 35 a comparação do fluxo de pacotes no dia do teste e também o de um dia anterior. O monitoramento do dia anterior, foi realizado pelo mesmo período de uma hora, no mesmo laboratório e pela mesma turma de usuários. A única diferença está no fato de que a rede estava estruturada na forma tradicional, não havia o controlador e nem o *switch OpenFlow*.

Nota-se que o fluxo de informações durante os sessenta minutos com a rede de estrutura tradicional e a rede definida por software, ficaram em torno de três gigabits.

## 5 RESULTADOS E DISCUSSÕES

Em nossos testes, o módulo L3-ARPSec se mostrou estável e cumpriu com os objetivos de mitigar o ataque MITM e controlar a troca de mensagens ARP na rede.

Na seção 3.3.1, os testes realizados utilizando o módulo L2\_learning no controlador da rede, foram suscetíveis ao ataque. Isto demonstrou a necessidade de criação de um módulo que não apenas gerenciasse de forma eficiente a troca de mensagens na rede, mas que também fosse seguro ao ataque MITM.

Com o módulo L3-ARPSec configurado no controlador de rede, testes foram executados na seção 4.5. Num primeiro momento, foram testadas as funcionalidades básicas, os *hosts* trocaram mensagens entre si provando que a distribuição de endereços MAC obteve sucesso.

Após isto, o *host* utilizado no cenário como atacante, desferiu ataques a fim de interceptar as mensagens entre os outros *hosts* da rede. O módulo L3-ARPSec conseguiu detectar os ataques e isolou temporariamente o *host* atacante. Tanto a função de detecção de inundação de ARPReply, quanto as varreduras nas tabelas ARPTTable e ARPTTableCandidate executadas em intervalos de tempos regulares e continuamente, se mostraram eficientes na detecção e reação ao ataque.

A função adaptativa de bloqueio do atacante se mostrou eficiente em casos de reincidência de ataques e como o módulo é programável facilmente, o valor definido inicialmente em dois pode ser ajustado pelo próprio administrador da rede.

Os testes realizados para validação do desempenho do módulo L3-ARPSec demonstraram não haver diferenças significativas no uso da rede *OpenFlow* se comparada a rede tradicional. Este fato foi importante para atestar que mesmo possuindo o mecanismo de segurança já mencionado, o módulo proposto não perdeu desempenho e executou todas as funcionalidades básicas de repasse e roteamento dos pacotes na rede.

Uma observação relevante está na possibilidade de um switch *OpenFlow* se conectar a um switch tradicional, formando uma rede híbrida. Entende-se assim, que as redes tradicionais podem ser migradas para o novo modelo de redes definidas por software, de maneira gradativa e sem significativo impacto estrutural.

## 6 CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

Nos capítulos iniciais deste trabalho foram brevemente apresentadas as características e funcionalidades das redes tradicionais e as redes definidas por software. Também foi apresentado o protocolo ARP e explicado seu funcionamento e importância em uma rede local de computadores. Ainda nestes capítulos foram demonstrados os ataques de segurança ao protocolo ARP, principalmente o ataque MITM que pode comprometer tanto redes locais tradicionais quanto redes locais definidas por software.

A possibilidade de programação no controlador SDN cria oportunidade para o desenvolvimento de técnicas e funcionalidades de gerenciamento dessas redes, foi o que ocorreu neste trabalho. No capítulo 4 foi apresentado o módulo L3-ARPSec, um conjunto de instruções que implementado no controlador de uma rede local SDN, controla a troca de mensagens ARP na rede e mitiga o ataque MITM.

Outros trabalhos sobre o assunto foram citados no capítulo 1, mas não foram encontrados nenhum que tratassem deste assunto em redes SDN e sim apenas em redes tradicionais.

Após executados diversos testes e simulações, conclui-se que o módulo proposto neste trabalho atingiu seus objetivos e representa uma linha de pesquisa vasta a ser explorada e melhorada ainda mais. Trabalhos futuros podem testar este módulo em redes com um número maior de elementos, executar testes de stress e reescreverem este módulo em outras linguagens de programação que suportem outros controladores SDN. Outros cenários também podem ser criados a fim de testar o modulo ainda mais e até mesmo ajustar as variáveis de tempo de punição, tempo de varredura nas tabelas e outras definidas no algoritmo,

Como todo software, o modulo L3-ARPSec pode evoluir e ser aperfeiçoado sempre que se desejar. Novas ideias e funcionalidades poderão ser agregadas ao projeto e contribuir para criar redes com melhor desempenho e mais seguras.

A partir deste trabalho foram submetidos e publicados alguns artigos em eventos internacionais e nacionais. Dois foram publicados e apresentados em Bogotá na Colômbia em junho de 2014, no “IEEE Colombian Conference on Communications and Computing” (COLCOM 2014) e outro foi submetido, aceito e será apresentado em Juiz de Fora, Minas Gerais, em setembro de 2015 no “XXXIII Simpósio Brasileiro de Telecomunicações”. Estes artigos estão na íntegra anexados a este trabalho no Apêndice C.

Alguns resultados obtidos neste trabalho ainda são inéditos, portanto, juntamente com futuras melhorias no módulo L3-ARPSec, poderão ser submetidos a periódicos e eventos.

## REFERÊNCIAS

BLUM, R. **Network performance toolkit: using open source testing tools**. Indiana: Wiley, 2003.

BRUSCHI, D.; OMAGHI, A.; ROSTI, E. S-ARP: a secure address resolution protocol. In: ANNUAL COMPUTER SECURITY APPLICATIONS CONFERENCE, 19., 2003, Las Vegas. **Proceedings...** Las Vegas: IEEE, 2003. p. 66-74.

CASADO, M.; FREEDMAN, M. J.; PETTIT, J.; LUO, J.; GUDE, N.; MCKEOWN, N.; SHENKER, S. Rethinking enterprise network control. **IEEE/ACM Transactions on Networking**, Urbana, v. 17, n. 4, p. 1270–1283, 2009.

ELLIOTT, C.; FALK, A. An update on the geni project. **SIGCOMM Computer Communication Review**, New York, v. 39, n. 3, p. 28–34, 2009.

GOUDA, M.G.; MOHAMED G.; HUANG, CHIN-TSER. A secure address resolution protocol. **International Journal of Computer and Telecommunications Networking, Computer Networks**, New York, v. 41, n.1, p. 57-71, 2003.

GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B.; CASADO, M.; MCKEOWN, N.; SHENKER, S. Nox: towards an operating system for networks. **Sigcomm Computer Communication Review**, New York, v. 38, n. 3, p. 105–110, 2008.

GUEDES, D; VIEIRA, L. F. M; VIEIRA, M. M; RODRIGUES, H; NUNES, R. V. Redes Definidas por Software: uma abordagem sistêmica para o desenvolvimento de pesquisas em Redes de Computadores. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 30., 2012, Ouro Preto. **Simpósio...** Ouro Preto: SBC, 2012. p. 161-212.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a internet: uma abordagem top down**. São Paulo: Pearson, 2010.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: ACM SIGCOMM WORKSHOP ON HOT TOPICS IN NETWORKS, 9., 2010, New York. **Proceedings...** New York: ACM, 2010. p. 19:1–19:6. Hotnets '10.

LOOTAH, W.; ENCK, W.; MCDANIEL, P. TARP: ticket-based address resolution protocol. In: ANNUAL COMPUTER SECURITY APPLICATIONS CONFERENCE, 21., 2005, Tucson. **Proceedings...** Tucson: IEEE, 2005. p. 116-125.

MARCONDES, C. **Projeto de desenvolvimento em *OpenFlow***: tutorial de *OpenFlow*. [S.l.: s.n.], 2011. Disponível em: <[http://www.inf.ufes.br/~magnos/IF/if\\_files/Tutorial.pdf](http://www.inf.ufes.br/~magnos/IF/if_files/Tutorial.pdf)>. Acesso em: 3 maio 2013.

MCKEOWN, N.; ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., TURNER, J. *Openflow*: enabling innovation in campus networks. **Sigcomm Computer Communication Review**, New York, v. 38, n. 4, p. 69–74, 2008.

OLIVEIRA, R. L. S.; SCHWEITZER, C. M.; SHINODA, A. A.; PRETE, L. R. Using mininet for emulation and prototyping software-defined networks. In: IEEE COLOMBIAN CONFERENCE ON COMMUNICATIONS AND COMPUTING- COLCOM, 1., 2014, Bogotá.. **Conference...** Bogotá: IEEE, 2014a. p. 1-6.

OLIVEIRA, R. L. S.; SCHWEITZER, C. M.; SHINODA, A. A.; PRETE, L. R. Simulation in an SDN network scenario using the POX controller. In: IEEE COLOMBIAN CONFERENCE ON COMMUNICATIONS AND COMPUTING- COLCOM, 1., 2014, Bogotá. **Conference...** Bogotá: IEEE, 2014b. p. 1-6.

PETERSON, L.; ROSCOE, T. The design principles of planetlab. **SIGOPS Operating Systems Review**, New York, v. 40, n. 1, p. 11–16, 2006.

PHILIP, R. **Securing wireless networks from ARP cache poisoning**. San Jose: [s.n.], 2007. (Master's Projects. Paper, 131). Disponível em: <[http://http://scholarworks.sjsu.edu/etd\\_projects/131](http://http://scholarworks.sjsu.edu/etd_projects/131)>. Acesso em: 26 ago. 2014.

TANENBAUM, A. S.; WETHERALL, D. **Redes de computadores**. 5. ed. São Paulo: Pearson Prentice Hall, 2011.

TENNENHOUSE, D. L.; WETHERALL, D. J. Towards an active network architecture. **Sigcomm Computer Communication Review**, New York, v. 37, n. 5, p. 81–94, 2007.

TRIPUNITARA, M.; DUTTA, P. A middleware approach to asynchronous and backward compatible detection and prevention of ARP cache poisoning. In: ANNUAL COMPUTER SECURITY APPLICATIONS CONFERENCE, 15., 1999, Phoenix. **Proceedings...** Phoenix: IEEE, 1999. p. 303-309.

TURNER, J. S. A proposed architecture for the geni backbone platform. In: IEEE SYMPOSIUM ON ARCHITECTURE FOR NETWORKING AND COMMUNICATIONS SYSTEMS,6., New York, 2006. **Proceedings...** New York: ANCS 2006. p. 1–10.

## APÊNDICES

### Apêndice A - Arquivo l2\_learning.py

#### l2\_learning.py

```
# Copyright 2011-2012 James McCauley
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at:
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""
An L2 learning switch.

It is derived from one written live for an SDN crash course.
It is somewhat similar to NOX's pyswitch in that it installs
exact-match rules for each flow.
"""

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpid_to_str
from pox.lib.util import str_to_bool
import time

log = core.getLogger()

# We don't want to flood immediately when a switch connects.
# Can be overridden on commandline.
_flood_delay = 0

class LearningSwitch (object):
    """
    The learning switch "brain" associated with a single OpenFlow switch.

    When we see a packet, we'd like to output it on a port which will
    eventually lead to the destination.  To accomplish this, we build a
    table that maps addresses to ports.

    We populate the table by observing traffic.  When we see a packet
```

from some source coming from some port, we know that source is out that port.

When we want to forward traffic, we look up the destination in our table. If we don't know the port, we simply send the message out all ports except the one it came in on. (In the presence of loops, this is bad!).

In short, our algorithm looks like this:

For each packet from the switch:

- 1) Use source address and switch port to update address/port table
- 2) Is transparent = False and either Ethertype is LLDP or the packet's destination address is a Bridge Filtered address?
  - Yes:
    - 2a) Drop packet -- don't forward link-local traffic (LLDP, 802.1x)
      - DONE
- 3) Is destination multicast?
  - Yes:
    - 3a) Flood the packet
      - DONE
- 4) Port for destination address in our address/port table?
  - No:
    - 4a) Flood the packet
      - DONE
- 5) Is output port the same as input port?
  - Yes:
    - 5a) Drop packet and similar ones for a while
- 6) Install flow table entry in the switch so that this flow goes out the appropriate port
  - 6a) Send the packet out appropriate port

```
"""
def __init__(self, connection, transparent):
    # Switch we'll be adding L2 learning switch capabilities to
    self.connection = connection
    self.transparent = transparent

    # Our table
    self.macToPort = {}

    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListener(self)

    # We just use this to know when to log a helpful message
    self.hold_down_expired = _flood_delay == 0

    #log.debug("Initializing LearningSwitch, transparent=%s",
    #          str(self.transparent))

def _handle_PacketIn (self, event):
```

```

"""
Handle packet in messages from the switch to implement above algorithm.
"""

packet = event.parsed

def flood (message = None):
    """ Floods the packet """
    msg = of.ofp_packet_out()
    if time.time() - self.connection.connect_time >= _flood_delay:
        # Only flood if we've been connected for a little while...

        if self.hold_down_expired is False:
            # Oh yes it is!
            self.hold_down_expired = True
            log.info("%s: Flood hold-down expired -- flooding",
                    dpid_to_str(event.dpid))

        if message is not None: log.debug(message)
        #log.debug("%i: flood %s -> %s", event.dpid, packet.src, packet.dst)
        # OFPP_FLOOD is optional; on some switches you may need to change
        # this to OFPP_ALL.
        msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
    else:
        pass
        #log.info("Holding down flood for %s", dpid_to_str(event.dpid))
    msg.data = event.ofp
    msg.in_port = event.port
    self.connection.send(msg)

def drop (duration = None):
    """
    Drops this packet and optionally installs a flow to continue
    dropping similar ones for a while
    """
    if duration is not None:
        if not isinstance(duration, tuple):
            duration = (duration, duration)
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet)
        msg.idle_timeout = duration[0]
        msg.hard_timeout = duration[1]
        msg.buffer_id = event.ofp.buffer_id
        self.connection.send(msg)
    elif event.ofp.buffer_id is not None:
        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        msg.in_port = event.port
        self.connection.send(msg)

self.macToPort[packet.src] = event.port # 1

```



```

if not self.transparent: # 2
    if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
        drop() # 2a
        return

if packet.dst.is_multicast:
    flood() # 3a
else:
    if packet.dst not in self.macToPort: # 4
        flood("Port for %s unknown -- flooding" % (packet.dst,)) # 4a
    else:
        port = self.macToPort[packet.dst]
        if port == event.port: # 5
            # 5a
            log.warning("Same port for packet from %s -> %s on %s.%s. Drop."
                % (packet.src, packet.dst, dpid_to_str(event.dpid), port))
            drop(10)
            return
        # 6
        log.debug("installing flow for %s.%i -> %s.%i" %
            (packet.src, event.port, packet.dst, port))
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet, event.port)
        msg.idle_timeout = 10
        msg.hard_timeout = 30
        msg.actions.append(of.ofp_action_output(port = port))
        msg.data = event.ofp # 6a
        self.connection.send(msg)

class l2_learning (object):
    """
    Waits for OpenFlow switches to connect and makes them learning switches.
    """
    def __init__ (self, transparent):
        core.openflow.addListeners(self)
        self.transparent = transparent

    def _handle_ConnectionUp (self, event):
        log.debug("Connection %s" % (event.connection,))
        LearningSwitch(event.connection, self.transparent)

def launch (transparent=False, hold_down=_flood_delay):
    """
    Starts an L2 learning switch.
    """
    try:
        global _flood_delay
        _flood_delay = int(str(hold_down), 10)

```

```
    assert _flood_delay >= 0
except:
    raise RuntimeError("Expected hold-down to be a number")

core.registerNew(l2_learning, str_to_bool(transparent))
```

## Apêndice B - Arquivo l3\_arpsec.py

### L3-arpsec.py

```
#
#código fonte do modulo L3-ARPSec
#
#Autor: Rogério Leão S. de Oliveira (leao.rogerio@yahoo.com.br)
#
#

from pox.core import core
import pox
log = core.getLogger()

from pox.lib.packet.ethernet import ethernet, ETHER_BROADCAST
from pox.lib.packet.ipv4 import ipv4
from pox.lib.packet.arp import arp
from pox.lib.addresses import IPAddr, EthAddr
from pox.lib.util import str_to_bool, dpidToStr
from pox.lib.recoco import Timer

import pox.openflow.libopenflow_01 as of

from pox.lib.revent import *

import time

# Timeout for flows
FLOW_IDLE_TIMEOUT = 20
FLOW_HARD_TIMEOUT = 60

#timeout for flows who blocking ArpAtacks
FLOW_IDLE_TIMEOUT_ATAACK = 0
FLOW_HARD_TIMEOUT_ATAACK = 2 * 60

# Timeout for ARPTable entries
ARP_TIMEOUT = 20

# timeout para ArpTableCandidate
ARP_CANDIDATE_TIMEOUT = 20

#timerCandidate - verifies the tables

TIMER_VERIF_CANDIDATE = 13
TIMER_VERIF_ARPTABLE = 8

#timerExpired
TIMER_EXPIRADA = 25
```

```

# Maximum number of packet to buffer on a switch for an unknown IP
MAX_BUFFERED_PER_IP = 5

# Maximum time to hang on to a buffer for an unknown IP in seconds
MAX_BUFFER_TIME = 5

class Entry (object):

    def __init__ (self, port, ip, mac, tp):
        self.timeout = time.time() + ARP_TIMEOUT
        self.port = port
        self.ip = ip
        self.mac = mac
        self.tp = tp

    def __eq__ (self, other):
        if type(other) == tuple:
            return (self.port,self.ip,self.mac,self.tp)==other
        else:
            return (self.port,self.ip,self.mac,self.tp)==(other.port,other.ip,other.mac,other.tp)
    def __ne__ (self, other):
        return not self.__eq__(other)

    def isExpired (self):
        if self.port == of.OFPP_NONE: return False
        if (self.tp == 0): return False    #fixo nao expira
        return time.time() > self.timeout

class EntryCandidate (object):
    def __init__ (self, port, ip, mac):
        self.timeout = time.time() + ARP_CANDIDATE_TIMEOUT
        self.port = port
        self.ip = ip
        self.mac = mac

    def __eq__ (self, other):
        if type(other) == tuple:
            return (self.port,self.ip,self.mac)==other
        else:
            return (self.port,self.ip,self.mac)==(other.port,other.ip,other.mac)
    def __ne__ (self, other):
        return not self.__eq__(other)
    def isExpired (self):
        if self.port == of.OFPP_NONE: return False
        return time.time() > self.timeout

class EntryArpFlood (object):
    def __init__ (self, port, qtde, firstupdate):
        self.firstupdate = firstupdate # primeira entrada
        self.port = port
        self.qtde = qtde

```

```

def __eq__ (self, other):
    if type(other) == tuple:
        return (self.port,self.qtde,self.firstupdate)==other
    else:
        return (self.port,self.qtde,self.firstupdate)==(other.port,other.qtde,other.firstupdate)
def __ne__ (self, other):
    return not self.__eq__(other)

def dpid_to_mac (dpid):
    return EthAddr("%012x" % (dpid & 0xfffffff),)

class l3_switch (EventMixin):
    def __init__ (self, fakeways = [], arp_for_unknowns = False):
        # These are "fake gateways" -- we'll answer ARPs for them with MAC
        # of the switch they're connected to.
        self.fakeways = set(fakeways)

        # If this is true and we see a packet for an unknown
        # host, we'll ARP for it.
        self.arp_for_unknowns = arp_for_unknowns

        # (IP,dpid) -> expire_time
        # We use this to keep from spamming ARPs
        self.outstanding_arps = {}

        # (IP,dpid) -> [(expire_time,buffer_id,in_port), ...]
        # These are buffers we've gotten at this datapath for this IP which
        # we can't deliver because we don't know where they go.
        self.lost_buffers = {}

        # For each switch, we map IP addresses to Entries
        self.arpTable = {}

        self.arpTableCandidate = {}

        self.arpTableFlood = {}

        # This timer handles expiring stuff
        # self._expire_timer = Timer(10, self._handle_expiration, recurring=True)

        Timer(TIMER_EXPIRADA, self.Verificar_Tuplas_Expiradas, recurring=True)
        Timer(TIMER_VERIF_CANDIDATE, self.Verificar_Candidate, recurring=True)
        Timer(TIMER_VERIF_ARPTABLE, self.Verificar_ArpTable, recurring=True)

        self.listenTo(core)

    def Verificar_Tuplas_Expiradas(self):
        for k in self.arpTable.keys():
            for k2 in self.arpTable[k].keys():
                if self.arpTable[k][k2].isExpired():

```

```

        del self.arpTable[k][k2]
        print "Entrada expirada, excluindo..."

    for k in self.arpTableCandidate.keys():
        for k2 in self.arpTableCandidate[k].keys():
            if self.arpTableCandidate[k][k2].isExpired():
                del self.arpTableCandidate[k][k2]
                print "Entrada candidate expirada, excluindo..."

def Verificar_Candidate(self):
    for k in self.arpTableCandidate.keys():
        for k2 in self.arpTableCandidate[k].values():
            if (self.Existe_Mac_na_ArpTable(k,k2.mac) == True):
                print "Ataque detectado, dois IPs para um MAC"
                self.Block_Temp_Arp_Flood(k,k2.ip,k2.mac,k2.port)
                continue
            if (self.Existe_Mac_na_ArpTableCandidate(k,k2.mac, k2.ip) == True):
                print "Ataque detectado, MITM"
                self.Block_Temp_Arp_Flood(k,k2.ip,k2.mac,k2.port)
                continue
            if k2.ip in self.arpTable[k]:
                if (self.arpTable[k][k2.ip].tp == 0):
                    print "Nao e possivel trocar tupla fixa"
                    continue
            print "Colocando candidate na arpTable, simples troca de porta, mac ou ip"
            del self.arpTable[k][k2.ip]
            self.Aprender_Ip_Mac(k,k2.ip,k2.mac,k2.port)

def Verificar_ArpTable(self):
    for k in self.arpTable.keys():
        for k2 in self.arpTable[k].values():
            if (self.Existe_Mais_de_um_Mac_na_ArpTable(k,k2.mac) == True):
                print "Ataque detectado, mais de um MAC para IPs distintos"
                self.Block_Temp_Arp_Flood(sdpid=k,mac=k2.mac)
                continue
            print "Ataque nao detectado para este MAC"

def Existe_Mais_de_um_Mac_na_ArpTable(self,dpid,mac):
    qt=0
    for v in self.arpTable[dpid].values():
        if (v.mac == mac and v.tp == 1):
            qt = qt + 1
            if (qt == 2):
                return True
    return False

def Existe_Mac_na_ArpTable(self,dpid,mac):
    for v in self.arpTable[dpid].values():
        if (v.mac == mac and v.tp < 2):
            return True
    return False

```

```

def Existe_Mac_na_ArpTableCandidate(self,dpid,mac,ip):
    for v in self.arpTableCandidate[dpid].values():
        if (v.mac == mac and v.ip <> ip):
            return True
    return False

def _imprimir_ArpTable (self, sdpid):
    print "Tabela ArpIP Switch: " + str(sdpid)
    print "IP          - MAC                - TYPE - INPORT"
    for subv in self.arpTable[sdpid].values():
        print str(subv.ip) + " - " + str(subv.mac) + " - " +str(subv.tp) + " - " +
str(subv.port)

    print "Tabela ArpIPCandidate Switch: " + str(sdpid)
    print "IP          - MAC                - INPORT"
    for subv in self.arpTableCandidate[sdpid].values():
        print str(subv.ip) + " - " + str(subv.mac) + " - " + str(subv.port)

def Aprender_Ip_Mac(self,sdpid,ip,mac,inport):
    if ip in self.arpTable[sdpid]:
        if (self.arpTable[sdpid][ip].tp == 0):
            print "%i %i %s IP origem ja existe permanente na tabela, ignorando..." %
(sdpid,inport,str(ip))
            elif (self.arpTable[sdpid][ip].mac == mac and self.arpTable[sdpid][ip].port == inport):
                print "%i %i %s IP origem ja existente na tabela, ignorando..." %
(sdpid,inport,str(ip))
            else:
                print "%i %i %s Alterado porta ou mac, enviando para table candidate..." %
(sdpid,inport,str(ip))
                if ip in self.arpTableCandidate[sdpid]:
                    print "%i %i %s Ja e candidate, ignorando..." % (sdpid,inport,str(ip))
                else:
                    self.arpTableCandidate[sdpid][ip] = EntryCandidate(inport, ip, mac)
                    print "%i %i %s Gravado em candidate..." % (sdpid,inport,str(ip))
            else:
                self.arpTable[sdpid][ip] = Entry(inport, ip, mac, 1)
                print ("%i %i %s %s IP learned..." % (sdpid,inport,str(ip),mac))
    self._imprimir_ArpTable(sdpid)

def Verificar_Mac_IP_Flood(self,sdpid,ip,mac,inport):
    if ip not in self.arpTableFlood[sdpid]:
        self.arpTableFlood[sdpid][ip] = {}
    if mac in self.arpTableFlood[sdpid][ip]:
        print "%i %s %s encontrado na tabela MacFlood, atualizando..." %
(sdpid,str(ip),str(mac))
        q = self.arpTableFlood[sdpid][ip][mac].qtde + 1
        self.arpTableFlood[sdpid][ip][mac].qtde = q
        if (q == 5):
            dif = time.time() - self.arpTableFlood[sdpid][ip][mac].firstupdate
            div = dif / q

```

```

    print "Diff: " + str(dif)
    print str(div)
    if (div < 3):
        print "Ataque detectado, flood Arp-Reply..."
        self.Block_Temp_Arp_Flood(sdpid,ip,mac,self.arpTableFlood[sdpid][ip][mac].port)
        del self.arpTableFlood[sdpid][ip][mac]
        return
    else:
        print "%i %s %s nao encontrado na tabela MacFlood, inserindo..." %
(sdpid,str(ip),str(mac))
        self.arpTableFlood[sdpid][ip][mac] = EntryArpFlood(inport, 1, time.time())
        print "Entrada ArpFlood - %i - %s - %s - %i - " %
(sdpid,str(ip),str(mac),self.arpTableFlood[sdpid][ip][mac].qtde)

def Block_Temp_Arp_Flood(self,sdpid,mac=None):

    """verifica quantidade de vezes que foi bloqueado e aumenta a punicao na reincidencia"""
    q = 1
    if mac in self.macTableBloqued[sdpid]:
        q = self.macTableBloqued[sdpid][mac].qtde + 1
        self.macTableBloqued[sdpid][mac].qtde = q
    else:
        self.macTableBloqued[sdpid][mac] = EntryMacBloqued(mac, 1)

    tempo_punicao = FLOW_HARD_TIMEOUT_ATAACK * q
    print ("%s punido %s veze(s), punicao...: %s" % (str(mac), str(q),str(tempo_punicao)))

    msg = of.ofp_flow_mod()
    msg.idle_timeout = FLOW_IDLE_TIMEOUT_ATAACK
    msg.hard_timeout = tempo_punicao
    msg.buffer_id = None
    msg.match.dl_src = mac
    core.openflow.getConnection(sdpid).send(msg) #envia flow de bloqueio

    for v in self.arpTable[sdpid].values():
        if (v.mac == mac and v.tp <> 0): # se encontrou o mac na tabela e nao tem status fixo
            del self.arpTable[sdpid][v.ip]

    for v in self.arpTableCandidate[sdpid].values():
        if (v.mac == mac): # se encontrou o mac na tabela e nao tem status fixo
            del self.arpTableCandidate[sdpid][v.ip]

def Find_Tuple(self,dpid,ip):
    if ip in self.arpTable[dpid]:
        if self.arpTable[dpid][ip].tp < 2:
            return True
        else:
            return False
    else:
        return False

```



```

def _handle_GoingUpEvent (self, event):
    self.listenTo(core.openflow)
    log.info("Up...")

def _handle_PacketIn (self, event):
    dpid = event.connection.dpid
    inport = event.port
    packet = event.parsed
    if not packet.parsed:
        log.info("%i %i ignoring unparsed packet", dpid, inport)
        return

    if dpid not in self.arpTable:
        # New switch -- create an empty table
        self.arpTable[dpid] = {}

    if dpid not in self.arpTableCandidate:
        self.arpTableCandidate[dpid] = {}

    if dpid not in self.arpTableFlood:
        self.arpTableFlood[dpid] = {}

    if packet.type == ethernet.LLDP_TYPE:
        # Ignore LLDP packets
        return

    if isinstance(packet.next, ipv4):          # se packet for IPv4
        print ("%i %i IP %s => %s" % (dpid,inport,
            packet.next.srcip,packet.next.dstip))

    # Learn or update port/MAC info
    self.Aprender_Ip_Mac(dpid, packet.next.srcip, packet.src, inport)

    # Tentar repassar o pacote
    dstaddr = packet.next.dstip
    if (self.Find_Tuple(dpid,dstaddr) == True):

        prt = self.arpTable[dpid][dstaddr].port
        mac = self.arpTable[dpid][dstaddr].mac
        if prt == inport:
            print ("nao repassar, portas in e out iguais...")
        else:
            print ("%i %i installing flow for %s => %s out port %i"
                % (dpid, inport, packet.next.srcip, dstaddr, prt))

        actions = []
        actions.append(of.ofp_action_dl_addr.set_dst(mac))
        actions.append(of.ofp_action_output(port = prt))
        match = of.ofp_match.from_packet(packet, inport)
        match.dl_src = None # Wildcard source MAC

```

```

msg = of.ofp_flow_mod(command=of.OFPFC_ADD,
                      idle_timeout=FLOW_IDLE_TIMEOUT,
                      hard_timeout=FLOW_HARD_TIMEOUT,
                      buffer_id=event.ofp.buffer_id,
                      actions=actions,
                      match=of.ofp_match.from_packet(packet,
                                                       inport))

event.connection.send(msg.pack())

elif isinstance(packet.next, arp):          # se packet for ARP
    a = packet.next

    if a.prototype == arp.PROTO_TYPE_IP:
        if a.hwtype == arp.HW_TYPE_ETHERNET:
            if a.protosrc != 0:

                # Learn or update port/MAC info
                self.Aprender_Ip_Mac(dpid,a.protosrc,packet.src,inport)

            if a.opcode == arp.REQUEST:      # se for um request de um host tenta responder
                # Maybe we can answer

                if (self.Find_Tuple(dpid,a.protodst) == True):    # busca na tabela
                    r = arp()
                    r.hwtype = a.hwtype
                    r.prototype = a.prototype
                    r.hwlen = a.hwlen
                    r.protolen = a.protolen
                    r.opcode = arp.REPLY
                    r.hwdst = a.hwsrc
                    r.protodst = a.protosrc
                    r.protosrc = a.protodst
                    r.hwsrc = self.arpTable[dpid][a.protodst].mac
                    e = ethernet(type=packet.type, src=dpid_to_mac(dpid), dst=a.hwsrc)
                    e.set_payload(r)
                    print ("%i %i Controller answering ARP directly for %s" % (dpid, inport,
                                     str(r.protosrc)))
                    msg = of.ofp_packet_out()
                    msg.data = e.pack()
                    msg.actions.append(of.ofp_action_output(port =
                                                            of.OFPP_IN_PORT))

                    msg.in_port = inport
                    event.connection.send(msg)
                    return

                print ("%i %i Nao encontrado na tabela, flooding ARP %s %s => %s" % (dpid,
inport,
                                     {arp.REQUEST:"request",arp.REPLY:"reply"}.get(a.opcode,
                                     'op:%i' % (a.opcode,)), str(a.protosrc), str(a.protodst)))

```

```

        msg = of.ofp_packet_out(in_port = inport, action = of.ofp_action_output(port =
of.OFPP_FLOOD))
        if event.ofp.buffer_id is of.NO_BUFFER:
            # Try sending the (probably incomplete) raw data
            msg.data = event.data
        else:
            msg.buffer_id = event.ofp.buffer_id
            event.connection.send(msg.pack())
    else:
        print ("%i %i nunca responder ARP diretamente %s" % (dpid, inport,
            str(a.protosrc)))
        self.Verificar_Mac_IP_Flood(dpid,a.protosrc,packet.src,inport)

def launch (fakeways="", arp_for_unknowns=None):
    fakeways = fakeways.replace(","," ").split()
    fakeways = [IPAddr(x) for x in fakeways]
    if arp_for_unknowns is None:
        arp_for_unknowns = len(fakeways) > 0
    else:
        arp_for_unknowns = str_to_bool(arp_for_unknowns)
    core.registerNew(l3_switch, fakeways, arp_for_unknowns)

```

### **Apêndice C – Artigos relacionados a este trabalho que foram submetidos e publicados**

1 - Using Mininet for Emulation and Prototyping Software-Defined Networks. IEEE COLCOM 2014.

2 - Simulation in an SDN network scenario using the POX Controller. IEEE COLCOM 2014.

3 - L3-ARPSec – A Secure *OpenFlow* Network Controller Module to control and protect the Address Resolution Protocol. SBRT 2015 (Simpósio Brasileiro de Telecomunicações).