UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
TECNÓLOGO EM REDES DE COMPUTADORES

MARIA ATRÍCIA SABINO MACIEL

AVALIAÇÃO DE EMULADORES PARA O ENSINO DE REDES DE COMPUTADORES

QUIXADÁ
2014

MARIA ATRÍCIA SABINO MACIEL

AVALIAÇÃO DE EMULADORES PARA O ENSINO DE REDES DE COMPUTADORES

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Tecnólogo em Redes de Computadores da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Tecnólogo.

Área de concentração: Computação

Orientador: Prof. Dr. Arthur de Castro Callado

QUIXADÁ
2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

M139a Maciel, Maria Atrícia Sabino
Avaliação de emuladores para o ensino de Redes de Computadores / Maria Atrícia Sabino Maciel. – 2014.
52 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Tecnologia em Redes de Computadores, Quixadá, 2014.
Orientação: Prof. Dr. Arthur de Castro Callado
Área de concentração: Computação

1. Emuladores (Programas de computador) 2. Redes de Computadores - Ensino
3. Rede de computador - Protocolos I. Título.

CDD 004.21

MARIA ATRÍCIA SABINO MACIEL

AVALIAÇÃO DE EMULADORES PARA O ENSINO DE REDES DE COMPUTADORES

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Tecnólogo em Redes de Computadores da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Tecnólogo.

Área de concentração: Computação

Aprovado em: 18 / Junho / 2014.

BANCA EXAMINADORA

_____
Prof. Dr. Arthur de Castro Callado (Orientador)
Universidade Federal do Ceará - UFC

_____
Prof. Dr. Danielo Gonçalves Gomes
Universidade Federal do Ceará - UFC

_____
Prof. Msc. Michel Sales Bonfim
Universidade Federal do Ceará - UFC

Aos meus pais, Valdeci Menezes, Alcilene Sabino, minha irmã Patrícia Sabino
e aos meus avós Francisco Sabino, Adalgisa Queiroz,
pelo apoio e incentivo em cada fase da minha vida.

AGRADECIMENTOS

Agradeço a Deus por me dar forças para suportar tudo e mais um pouco, pelas lições que aprendi, pelas experiências que vivi.

Ao professor Arthur, pela orientação desse trabalho, por sua disposição em ajudar e seu grande profissionalismo.

Ao professor Samy, pela oportunidade de iniciação à pesquisa, por sua dedicação em sempre nos auxiliar academicamente. Ao amigo Samy, o responsável por grandes mudanças na minha vida.

Aos colegas que viraram amigos na UFC, pela ajuda e a amizade. Luclécia Lopes, Eudes Sousa, Edigleison Barbosa, Clycia Najara, Janael Pinheiro, Matheus Medeiros, Rejane Vasconcelos, Felipe Alex, Joel Pereira, Tallis Maia, Dani Dutra, Sheila Alves, Cidney Falcão e aos demais colegas do curso de redes de computadores.

Aos colegas e amigos do PET, no qual vivenciei grandes momentos e aprendizado. Letícia Fernandes, Thalita Maria, Paula Santos, Evelyne Avelino, Glailton Costa, Rogério Carvalho, Alexsandro Gondim, Anderson Gonçalves, Marcos Ribeiro, Priscila Rodrigues, Otávio Augusto.

Aos professores David Senna, Tânia Pinheiro, Ricardo Reis, Diana Braga, João Marcelo, e Jeandro Bezerra, pela contribuição em projetos e atividades da minha vida acadêmica.

Ao Zarathon Maia pelas grandes ajudas, a quem terei uma gratidão eterna, aos servidores Jones Bezerra, Flávio, Bismack, Kaká, Venício Oliveira e Maurício Nogueira.

Aos meus amigos e familiares por fazerem parte da minha vida. E a todos que contribuíram de alguma forma para meu crescimento pessoal, me tornando assim uma pessoa melhor.

Aprendizagem é estar ciente de que nunca se sabe tudo.
(Atrícia Sabino)

RESUMO

A emulação de redes é uma técnica útil para o estudo e avaliação de desempenho de aplicações. O uso de emuladores de redes é de fundamental importância no processo de novas pesquisas, desenvolvimento de novos protocolos de redes, análises de desempenho de uma rede, como também no desenvolvimento de novas tecnologias. Esse trabalho tem como finalidade geral introduzir essa técnica no ensino de redes de computadores e avaliar emuladores para uso no ensino. Este trabalho concentra-se em três emuladores disponíveis gratuitamente: *Core*, *Netkit* e o *Mininet*. Avaliamos suas características e diferenças, discutimos algumas de suas métricas, para proporcionar uma visão ampla das diferentes ferramentas de emulação de redes.

Palavras chave: Emulador, Redes de Computadores, Ensino.

ABSTRACT

The network emulation is a useful aproach for the study and performance evaluation of technical applications. The use of network emulators has fundamental importance for new research, development of new network protocols, performance analysis of a network process, but also in developing new technologies. This work has as main purpose to introduce this technique in the teaching of computer networks and evaluate emulators for use in teaching. This paper focuses on three freely available emulators: Core, Netkit and Mininet. We evaluate its features and differences, discuss some metrics to provide a broad overview of the different tools for network emulation.

Keywords: Emulator, Computer Network, Teaching.

Lista de Ilustrações

Lista de tabelas

SUMÁRIO

1 INTRODUÇÃO

A grande rede mundial de computadores, a Internet, continua a crescer, fazendo com que seu ambiente seja crucial para o desenvolvimento, criação de novos protocolos e aplicações. Testar esses novos protocolos e aplicações em laboratório de rede imitando condições reais ainda é possível, mas pode haver dificuldades nas condições de ambiente e controle de tempo. (KIDDLE, 2004)

Para isso existem diferentes tipos de técnicas que permitem a implementação de experimentos que depende da infraestrutura e da sua necessidade. Os três métodos mais conhecidos são: testes em rede real, simulação de redes e o método de emulação de rede. A simulação de rede permite, através da imitação de resultados (e não do processo avaliado em si), fazendo com que as condições e os elementos de uma rede sejam controlados e repetíveis, em cada execução. A técnica de emulação também permite controlar e repetir vários experimentos, no entanto isso é feito em tempo real. (SARI; WIRYA, 2007).

O uso da técnica conhecida como emulação, consiste da execução de uma aplicação em um ambiente sintético (simulado), podendo reproduzir diversas vezes uma configuração com baixo custo, modificando a infraestrutura. A camada de emulação é configurada para emular latência, taxa de transmissão, dentre outras métricas de redes. (NUSSBAUM, 2009).

Dentre os objetivos da emulação, Martins (2011) destaca que reduzir custos para otimizar os testes relacionados ao desenvolvimento de *softwares* de redes são pontos que devem ser ressaltados ao falar da técnica de emulação. Há ainda a vantagem de executar aplicativos de máquinas reais, em pequenos cenários comparados aos de uma rede real. O autor ainda complementa descrevendo como seria uma aplicação usando a técnica da emulação.

> Através da simulação das ligações entre os nós, o que implica geralmente uma máquina para representar muitos nós em uma topologia. Emulação é um meio termo entre simulação e bancos de ensaio, mantendo um ambiente flexível em termos de dimensão e parametrização, mas também oferece a confiabilidade de um testbed e redução dos custos de implantação. (Martins, 2011, p. tradução livre).

Utilizando-se dessa técnica é possível criar e gerenciar vários perfis de comportamentos de uma rede de computadores. Nas definições de Ferreira et al (2005), a emulação consiste na utilização de um *software* especial, designado por emulador, de forma a reproduzir o comportamento de uma plataforma que pode ser tanto um *hardware* ou um *software*.

O uso de emuladores de redes é de fundamental importância no processo de novas pesquisas, desenvolvimento de novos protocolos de redes, análises de desempenho de uma rede, como também no desenvolvimento de novas tecnologias. Com essas possibilidades, testes de cenários podem ser feito de forma mais realista e controlável. Para tal o emulador é capaz de permitir a construção de cenários na emulação, normalmente uma rede real possui diversos perfis de comportamentos, o que dificulta identificar e isolar problemas. Usando emuladores é possível criar esses perfis e ainda repetir um experimento, diversas vezes, mantendo o comportamento da rede inalterado. Além disso, com o uso dessas ferramentas, os parâmetros que compõem o comportamento da rede podem ser controlados individualmente, permitindo então avaliar o impacto de cada um desses fatores separadamente. (KROPOTOFF, 2002).

Partindo da necessidade de conhecer e escolher um emulador para então executar a tarefa de emulação, no qual possa ser utilizado nas disciplinas do curso de Redes. Faz-se necessário então, ao invés de aplicar apenas aulas teóricas expositivas ou ainda aulas práticas que pouco refletem o que um administrador de redes poderá encontrar no seu dia a dia, encontrar uma maneira de complementar essa aprendizagem com atividades que possam agregar conhecimentos necessários para sua vida profissional. Baseado nesse contexto percebe-se que o computador é um instrumento de eficácia indiscutível ao processo de ensino, tornando-se um fator predominante no processo de aprendizagem em diversas áreas do conhecimento (GADELHA, et al. 2010).

Na concepção de alguns autores, existe a afirmação de que no estudo de ciências físicas, o uso de simuladores torna-se frequente para auxiliar e reforçar a fixação de conteúdos por partes dos alunos, tendo em vista que recorrer as experiências reais é uma tarefa difícil de reproduzir ou observar (FIOLHAS e TRINDADE 2003). Trazendo essa percepção para o ensino de redes de computadores, tem-se como solução o uso de emuladores de redes que vai além da simulação ressaltada pelos autores. Dessa forma, é possível reproduzir um cenário de estudo com alto índice de complexidade, resultando em uma proximidade a ambientes reais.

Esse trabalho testou e comparou emuladores, sendo possível detectar ou eleger um destes para uso dedicado em disciplinas do curso de graduação na área de redes de computadores. Além disso, foram descritas as funcionalidades, vantagens e problemas de cada emulador. A comunidade estudantil, bem como profissionais em geral terão um ganho substancial, visto que no campus da UFC em Quixadá ainda não foram desenvolvidas pesquisas à respeito desse tema.

1.1 Contexto e Motivação

Com o intuito de contribuir com a comunidade acadêmica da Universidade Federal do Ceará no campus de Quixadá, esse trabalho visa selecionar entre as ferramentas *open source*, ou seja, com o código aberto, aquela que venha a se encaixar melhor para o uso de emulação no ensino de redes de computadores. Entre as ferramentas disponíveis para emulação de redes, e as mais utilizadas pela academia existem algumas com características marcantes e de código aberto. Estas serão avaliadas e comparadas a fim de descobrir suas limitações e aspectos positivos.

Já existem várias soluções e ferramentas para emulação de rede. No entanto, as características e funções dessas ferramentas ainda não foram comparadas para uso específico no ensino de redes, apesar de serem amplamente utilizado pela comunidade acadêmica em pesquisas. Este trabalho contribui com um estudo comparativo de emuladores de redes, destacando suas diferenças, vantagens e características.

1.2 Objetivos

O objetivo geral desse trabalho é avaliar ferramentas de emulação para uso no ensino de disciplinas do curso de redes de computadores.

Os seguintes objetivos específicos foram levados em conta para alcançar a meta do objetivo geral.

- Identificar emuladores para análise;
- Avaliar a eficiência de diferentes emuladores;
- Comparar e recomendar emuladores de redes;

1.3 Metodologia

Esta seção tem por objetivo detalhar os passos para avaliação e comparação dos emuladores utilizados para a realização desse trabalho. Primeiramente fez-se necessário:

1. Pré- Selecionar emuladores *open source* e fazer um estudo de caso sobre emulação no ensino. Nessa etapa selecionou-se três ferramentas de código aberto e utilizadas pela academia.

2. Criar um cenário geral de estudo, criando uma padronização e com isto as ferramentas foram testadas e comparadas em cima desse cenário.

3. Para cada ferramenta observou-se: funcionalidades da ferramenta, configurações e suas características principais.

4. Executar os experimentos e coletar os critérios de avaliação, nessa etapa é possível observar o desempenho de aplicações na rede emulada.

5. Avaliação e comparação dos emuladores. A avaliação permitiu a comparação das ferramentas estudadas, a partir da qual pudemos tirar conclusões e propor a adoção de uma destas ferramentas para uso nas disciplinas e atividades de avaliação de desempenho em redes.

1.4 Organização do Texto

Essa seção tem por objetivo detalhar a organização do texto, depois da seção **1** de introdução no qual é mostrado o objetivo principal do trabalho, contexto e motivação. A seção **2** mostra os principais conceitos, e um levantamento bibliográfico sobre emulação, emuladores e experimentos, além de uma breve descrição dos emuladores selecionados. Já na seção **3** temos os resultados obtidos com a execução dos experimentos, em que é feito uma avaliação de cada emulador. Finalizando com a seção **4** onde são descritos as conclusões e considerações obtidas com a avaliação dos emuladores.

2 REVISÃO BIBLIOGRÁFICA

Essa seção tem por objetivo detalhar o conceito de emulação, sua real funcionalidade na área de redes de computadores. Na seção 2.1 teremos uma breve descrição dos benefícios dessa técnica para o ensino de redes. Em seguida na seção 2.2 será discutido

sobre os emuladores avaliados no trabalho, para então abordar o conceito e aplicação de experimento, e possíveis métricas a serem avaliadas em um experimento.

2.1 Emulação em Redes de Computadores

Para desenvolvimento, estudos, experimentos e testes de novos protocolos em uma rede, existem três métodos que podem ser usados por cientistas para testes de suas aplicações em uma rede de computador especifica, são eles teste em rede real, técnica de simulação em redes e o método de emulação em rede. Cada método tem seus pontos fortes e fracos no qual podem ser usados em conjuntos para suprir as necessidades uns dos outros. (SARI, WIRYA, 2007).

Abordagens atuais sobre emulação de redes oferecem maneiras flexíveis para imitar funcionalidades de redes de computadores. Herrscher e Rothermel (2002), identificaram três camadas de abstração que podem definir a emulação. Perante essas abstrações observam-se que as três camadas relatadas pelos autores são a camada de transporte, camada de rede e camada de enlace. Suas funcionalidades principais são descritas a seguir.

A camada de transporte reproduz algumas características do canal de comunicações responsáveis pelas trocas de informações entre os processos em execuções. A de rede tem a capacidade de aproximar-se do comportamento real, entre eles a comunicação fim-a-fim entre *hosts*, e nessas comunicações alguns dos comportamentos que podem ser imitados, destacando-se a perda e o atraso de pacotes. Finalmente, na camada de enlace, característica principal é a emulação de links de redes individuais, como por exemplo, taxa de transmissão e atraso (HERRSCHER; ROTHERMEL, 2002).

Para demonstrar o funcionamento da emulação é preciso utilizar um software, conhecido como emulador. Ele é responsável por emular o comportamento de uma plataforma, seja de hardware ou software. A vantagem principal desse tipo de abordagem é o elevado grau de proximidade que se pode chegar das funcionalidades de um objeto original (LEE et al. 2002; ROTHENBERG et al. 1999).

2.2 Emuladores

Emuladores serão os programas capazes de executar os mesmos programas de um sistema real sobre outro sistema hospedeiro e também reproduzem os mesmos resultados para entradas iguais. Estes programas criam uma camada de *software* entre as plataformas hóspedes e hospedeiras. (CARARISSIMI, et al. 2008).

Em linguagem computacional, o emulador seria como a interpretação de um código. Com isso o emulador pode "enganar" as plataformas fazendo com que as operações das máquinas sejam implementadas em um *software*. Laureano et al, (2008) reafirma que emuladores são complexos, isso acontece porque além de simular quase todas as instruções do processador ainda necessita simular características que estejam presentes no *hardware.*

A seguir são apresentados diferentes emuladores pré-selecionados para avaliação neste trabalho. Não descartamos a possibilidade de outros serem identificados e considerados posteriormente.

2.2.1 Core Emulator

O primeiro emulador considerado para início das avaliações foi o *Core* (Common Open Research Emulator), é uma ferramenta de emulação de rede *Open Source.* O *Core* é usado em sistemas operacionais *linux* e pode emular PCs, roteadores, e *links* de redes. Essa ferramenta é baseada no código livre do I*ntegrated Multi-protocol Network Emulator/Simulator* (*IMUNES*) da Universidade de Zabred. Esse emulador possui suporte a redes *wireless*, *scripts* de mobilidade, IPsec, IPv6, emulação distribuída, controle de roteadores externos linux, uma API remota e *widgets* gráficos. (AHRENHOLZ et al., 2008)

O *Core* possui um *daemon* (*backend*) que é responsável por gerenciar as sessões de emulações. Basicamente seu funcionamento consiste em conectar nós e redes via interfaces de cada nó, e então o *daemon* responsável pela criação da sessão é executado via interface gráfica *CORE GUI* (*frontend*). Esses *daemons* usam módulos *python* que são importados diretamente de *scripts python* da instalação. É usado com mais frequência para emular nós em uma máquina real, e seu foco é mais voltado para emulação na camada 3, testes de protocolos e aplicações de *software*.

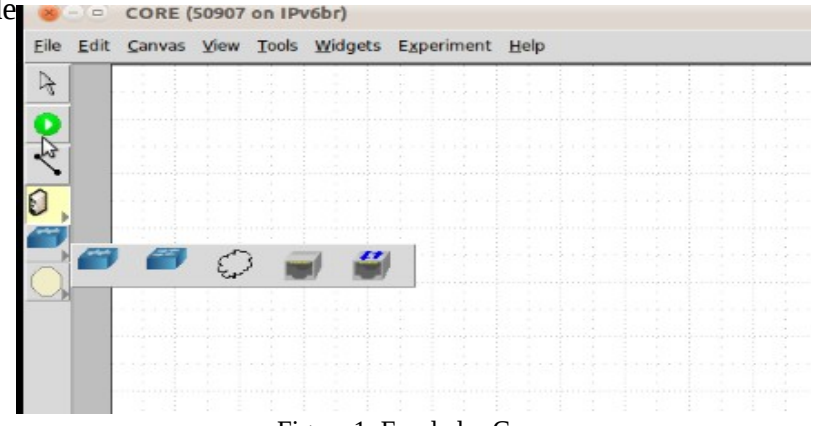

Figura 1: Emulador Core.

2.2.2 NetKit Emulator

Na sequência da avaliação de ferramentas o segundo emulador testado foi o *Netkit.* O *software* de código livre permite a execução e gerenciamento de experimentos em redes de computadores, incluindo ainda *hardware* para seu suporte e interligação com ativos reais. O *Netkit* foi desenvolvido pela *Roma Tre Universitá* na Itália, e logo recebeu o apoio da comunidade científica. (RIMODINI 2007).

O *Netkit* dispõe de um conjunto de ferramentas com códigos abertos licenciado pela GPL. O experimento criado no emulador pode ser iniciado e criado através de scripts ou através da linguagem *NetMl*, que é uma linguagem baseada em XML para redes. As máquinas virtuais criadas no *Netkit* é um computador completo rodando uma distribuição Debian em modo usuário. A figura a seguir mostra as máquinas sendo representada por terminais.

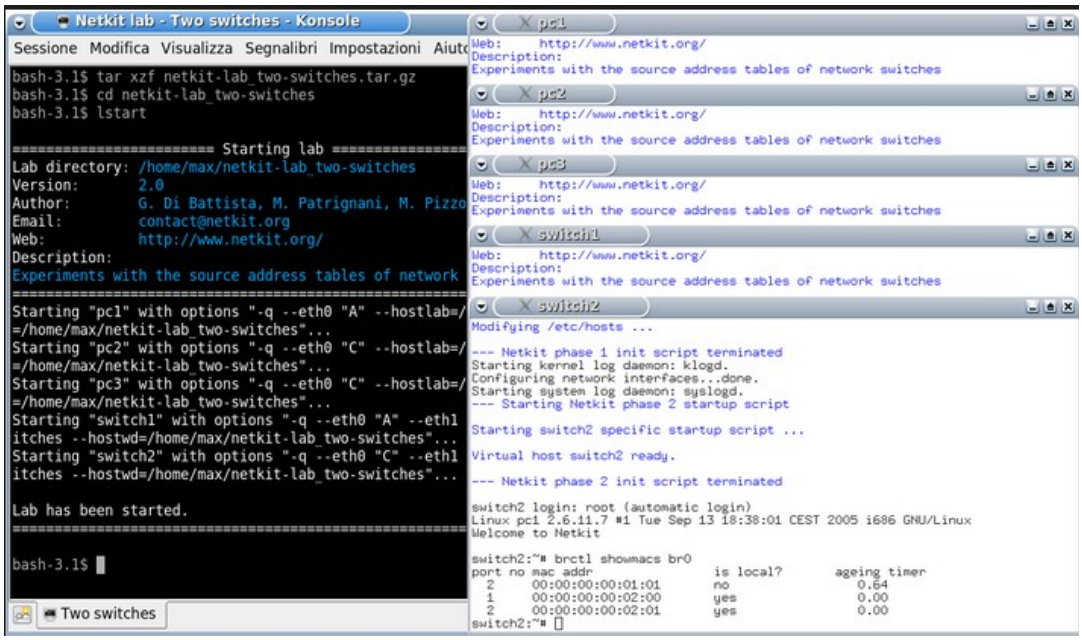

Figura 2: Máquinas criadas no Netkit.

Dos recursos que podem ser testados com o *Netkit*, podemos destacar; protocolos em todas as camadas, protocolos de roteamento, comutação de pacotes por MPLS, ferramentas de segurança, detecção de intrusão, encapsulamento, filtro de pacotes, montagem de pacotes. Podem ser utilizadas linguagens de *script awk, bash, expect, python.* (GURGEL; BARBOSA; BRANCO, 2012)

2.2.3 Mininet Emulator

O *Mininet* é um emulador criado com o objetivo de criar redes definidas por software (*Software Defined Networks – SDN*). Redes escaláveis em um único computador usando processos em redes distintas. Com isso o usuário cria, interage, modifica e compartilha um protótipo de SDN, fornecendo uma maneira mais fácil para rodar no hardware. (GOMES;

MADEIRA, 2012). Nesse trabalho, ele foi usado com o intuito de emular apenas uma rede comum, com o *switch* modo usuário. A figura a seguir demonstra como é o funcionamento do *Mininet.*

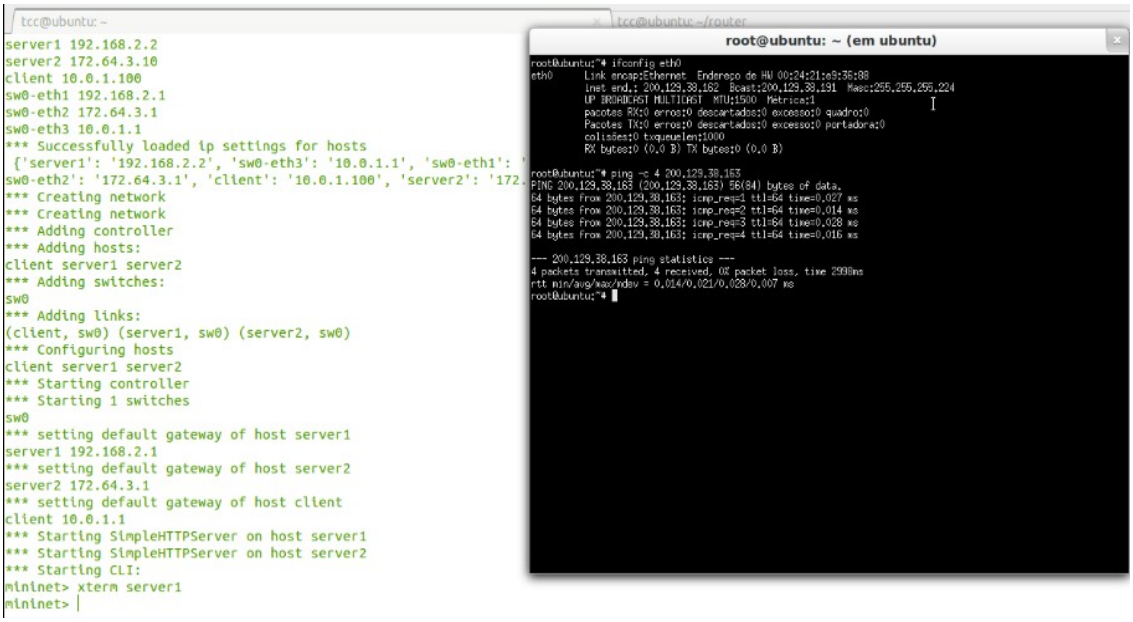

Figura 3: Máquina executando no Mininet.
Fonte: elaborada pelo autor.

O *Mininet* é um emulador que tem uma configuração simples, considerando que você o uso como máquina virtual ou utilize um *script* disponibilizado para *Debian* e *Ubuntu* que executa uma configuração automática. Com um simples comando é criado o *controlador, os switches e* os *hosts*.

2.3 Experimentos no Ensino de Redes de Computadores

Podemos imaginar redes de computadores como sendo dois ou mais nós em uma rede que usam protocolos em comum para se comunicarem. Tanenbaum (2003) define redes de computadores como sendo um conjunto de computadores independentes capazes de trocar informações.

A partir da criação de um cenário, torna-se viável verificar o controle da rede e seus recursos. A possibilidade de poder fazer testes e implementações na emulação oferece a pesquisadores a capacidade de validar suas experiências, sem a necessidade do uso de uma rede real e com a vantagem de permitir controle e repetibilidade. Essas experiências podem

ser testes com largura de banda, atrasos de rede, tempo de resposta de um serviço a determinada requisição. Elementos reais podem completar as aplicações emuladas, nas palavras de Guruprasad et al. (2005).

> Tais como hospedeiros finais e protocolo de implementações sintéticas, simulados, ou elementos abstraídos, como as ligações de rede, nós intermediários e de tráfego de segundo plano. Elementos reais são parcialmente ou totalmente simulados várias vezes, de maneiras diferentes, dependendo das necessidades do experimentador e dos recursos disponíveis. (Guruprasad et al, 2005, tradução livre).

No método científico, um experimento é válido se for observável, repetível, controlável e falseável. No contexto de ciência da computação a repetição e o controle de experimentos requer acesso livre a todos os códigos, scripts e dados que venham a ser usados para reproduzir os resultados. (HANDIGOL et al. 2012).

2.3.1 Métricas de Redes

Nesse trabalho adotou-se algumas métricas de redes, como Taxa de Transmissão, Perda de Pacotes e Latência. A definição de métrica depende do que será avaliado no desempenho da aplicação. Porém, o objetivo poder ser apenas para avaliar o desempenho de uma rede.

A *latência* é a medida do tempo que um pacote leva desde sua origem até o destino, esse tempo é calculado com a soma dos atrasos de processamento e o atraso de propagação ao longo da transmissão.

A *largura de banda* é conceituada como sendo a quantidade máxima de dados que podem ser transportados da origem até o destino. Em geral ela é muito associada ao desempenho da rede. É uma métrica muito importante quando se avalia a qualidade dos serviços de uma rede.

A métrica de *perda de pacotes* avalia o número de dados que foram transmitidos pelo transmissor e que não foram recebidos em seu destino. Mais conhecido como taxa de perda de pacotes, quando tomada na razão com o total de pacotes transmitidos. (COSTA, 2008)

3 EXPERIMENTOS E RESULTADOS

Nessa seção serão apresentados os resultados colhidos e analisados de cada emulador. Com base na execução do cenário definido para o trabalho.

3.1 Configuração do Experimento

Nesta seção, são mostrados os requisitos para a realização do experimento, como também como foi o desempenho de cada emulador. Os resultados, características e discussões resultantes do experimento também são mostrados.

A configuração do cenário foi feita com duas máquinas físicas e duas emuladas. Utilizou-se uma máquina real executando o *ubuntu server* 12.04 , com o emulador instalado, emulando *hosts* e outra máquina real com o *ubuntu* 12.04 *desktop*. Permitindo então que máquinas emuladas pudessem se comunicar com uma física. A figura mostra como foi arquitetado o cenário.

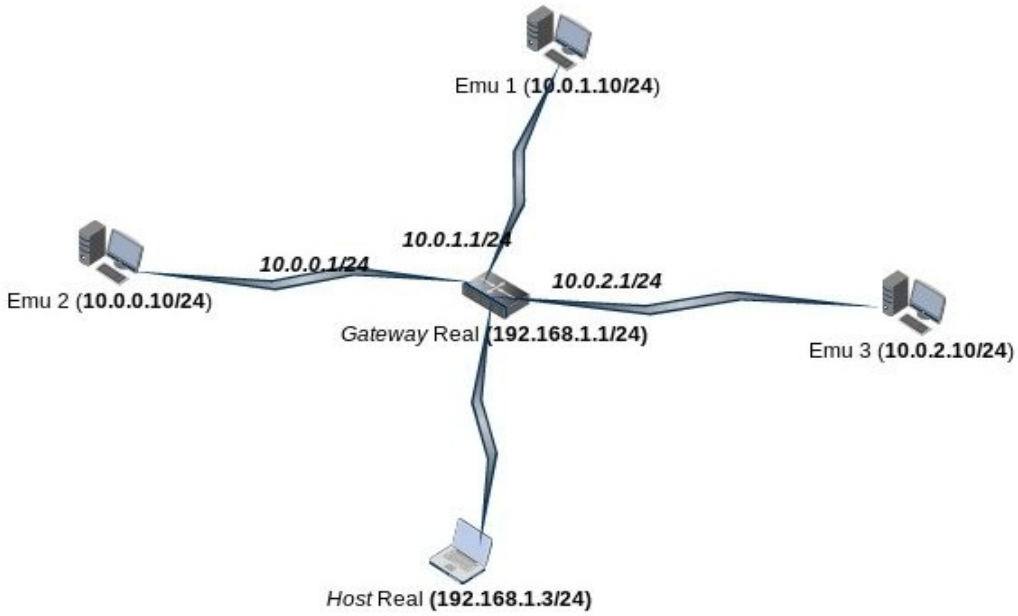

Figura 4: Cenário Experimental

Esse cenário funciona da seguinte forma, uma máquina real *linux* com o emulador instalado, fazendo o papel de *gateway* conectada a uma um física através da placa *eth0* (Rj45). A máquina *gateway* é configurada para rotear o encaminhamento de pacotes. Em cada máquina é preciso adicionar rotas para que as redes possam ser enxergadas, permitindo assim ser feitos os testes entre ambiente real com emulado e, então, coletar as métricas e critérios tidos como objetivos nesse trabalho.

3.2 Resultados

Para que se tenha um equilíbrio entre o desempenho de uma rede e o ensino baseado em emulação, definiram-se alguns critérios que foram levados em conta na comparação dos emuladores. Dentre eles, selecionamos para atender ao ensino configuração/instalação do emulador, facilidade de uso, disponibilidade operacional; e para o desempenho de redes largura de banda, latência, perda de pacotes.

3.2.1 Avaliação Core

Para instalar e executar o emulador *Core*, é necessário qualquer máquina com sistema operacional linux, devido ao fato do emulador criar várias máquias virtuais é recomendado utilizar uma máquina com mais memória RAM e CPU. Em geral a recomendação deve ser:

- ✓ 2GB ou mais de memória RAM
- ✓ 2.0GHz ou um processador x86
- ✓ x11 para interface gráfica e SSH

Todos os testes e execuções da emulação foram feitos no sistema operacional linux, Ubuntu 12.04 LTS. Iniciamos nosso experimento com a ferramenta *Core Emulator*. Esse emulador pode ser usado via GUI ou com *Script* em *python.* Nesse trabalho optamos por usar a interface gráfica, em anexo 46 também temos como criar uma topologia em código *python.*

Tabela 1 Critérios e Avaliações

Critérios	***Core Emulator***	***Netkit Emulator***	***Mininet Emulator***
Configuração/Instalação do Emulador	Fácil	Fácil / Necessário Conhecimento Linux	Intermediário / Necessário Conhecimento Linux
Facilidade de Uso	Fácil	Fácil	Fácil
Serviços de Rede Suportados	Ipv6, Ipsec, OSPFv3, DHCPv6, DNS, ICMPv6, Samba, Redes *Wireless.*	Ipv6, NAT, 802.1Q , *Radius*, 802.1d *Bridging* , *Spanning Tree*	*OpenFlow*, NAT, SSH, Redes Wireless, DHCP
Configuração da Taxa de Transmissão	sim	sim	sim
Latência	sim	sim	sim
Perda de Pacotes	sim	sim	sim
Disponibilidade (Plataforma Operacional)	- FreeBSD - Linux Disponível para *kernel* > 2.6.24	-Linux - Disponível para todos os *Kernels*	-Linux Versão 2.0.0 para *Ubuntu* 12.04+

Na configuração e instalação da ferramenta *Core*, pode-se notar que existe uma facilidade para realizar essa tarefa. Para sua instalação no *ubuntu* é necessário apenas instalar alguns pacotes que são pré-requisitos, e então partir para a instalação das bibliotecas e do módulo para roteamento. Para concluir a instalação, basta iniciar o *daemon* e usar o *Core*. Pensando na facilidade de uso optamos por usar a interface gráfica da ferramenta *Core GUI*,

simples e intuitiva. Basta conhecer alguns objetos, e barras de configurações e então é possível usar o *Core* facilmente. Com uma simples ação de selecionar e arrastar os ativos de rede é possível montar um cenário de rede rapidamente. A parte mais trabalhosa é a de habilitar alguns serviços que raramente falham e a parte de roteamento.

De acordo com a documentação oficial o *Core Emulator* está disponível para plataformas operacionais *Linux* e *FreeBSD*, no linux a partir do *kernel* 2.6.24 e no FreeBSD a partir de *kernels* 4.11. A ferramenta também pode ser usada virtualmente, inclusive possui algumas máquinas prontas para uso e testes do protocolo IPv6.

Depois de configurar uma máquina real para fazer o papel de *gateway* e uma máquina física para a comunicação com as máquinas emuladas, utilizou-se a ferramenta *iperf*, comumente usada para medir o desempenho de redes ao injetar tráfego na rede, permitindo coletar os dados das máquinas emuladas. Ao utilizar a ferramenta "Ping" entre uma máquina física e uma máquina emulada no *Core* foi possível saber qual a latência 0.691 ms, perda de pacotes 112 pacotes e taxa de transmissão 92 Mbits/sec.

3.2.2 Avaliação Netkit

A segunda ferramenta escolhida para ser avaliada foi o *Netkit emulator*, seu funcionamento consiste no uso de dois componentes: um conjunto de scripts que auxiliam a criação de cenários e um conjunto de dispositivos virtuais. Com uma instalação fácil ou até mesmo o uso a partir de um *live cd*. Alguns conhecimentos básicos do sistema operacional linux são suficientes para instalá-lo e configurá-lo.

Disponível apenas para plataformas operacionais linux baseadas em Debian, o *Netki*t também só cria máquinas linux emuladas. Ainda não existem versões para o Windows, uma alternativa para que usa *Windows* é utilizá-lo emulado em uma máquina virtual ou para facilitar mais ainda rodar a versão *live cd*. Em geral os principais requisitos de hardware para instalação e funcionamento são:

- ✓ Mínimo de 300MB de memória RAM.
- ✓ Necessário um bom espaço em disco, visto que cada máquina virtual criada tem um tamanho de 10GB.
- ✓ Sistema de Arquivo ext2/ext3

Na configuração é necessário criar algumas variáveis de ambientes que são exigidas para que o emulador funcione corretamente. Uma característica negativa da ferramenta é que sempre que a máquina for desligada, ela vai perder as configurações, exigindo que o usuário configure tudo novamente, mas como as variáveis são para o *bash*, basta apenas escrever os **export** no arquivo **.bash_rc** dessa forma ele será iniciado juntamente com o sistema.

O próximo passo da avaliação foi coletar algumas métricas da rede emulada com o *Netkit*, seguindo o cenário global definido na seção **4.1**. Mais uma vez fez-se uso do *iperf* para inserir tráfego na rede. Outro ponto negativo da ferramenta é a tarefa de adicionar internet nas máquinas virtuais emuladas, por se tratar de máquinas virtuais é preciso ainda criar as pontes para a internet fosse compartilhada, surgiu essa necessidade devido ao fato que cada *host* tivesse o *iperf* instalado para então comunicar com a máquina real.

A métrica de perda de pacote foi feita diminuindo a capacidade do enlace da rede, o que possibilita ver qual é a perda de pacotes na comunicação da rede real com a virtual. No total de pacotes transmitidos a perda foi de 154 pacotes. Também foi possível medir a largura de banda da transmissão equivalente a 89,8 Mbits/sec e a latência de 0.442 ms. Percebe-se que até agora os emuladores conseguem medir essas métricas, contribuindo para suas características positivas para o ensino.

3.2.3 Avaliação Mininet

O último emulador testado foi o *mininet*, por usar um único *kernel* do *Linux* para criar todos os *hosts* emulados, o que significa que não é possível executar *softwares* dependentes do Windows ou outros sistemas operacionais, embora seja possível executar em uma máquina virtual (VM). Com uma instalação relativamente fácil no *ubuntu*, em uma máquina real basta apenas digitar o comando ***apt-get instal mininet*** e com isso a ferramenta é instalada facilmente. Em geral as recomendações de requisitos de hardware são:

- ✓ 1GB de memória RAM
- ✓ Pelo menos 5GB de espaço livre em disco
- ✓ x11 *Server* para gerenciamento das máquinas virtuais.

Já sua configuração não é tão trivial, visto que com apenas esse comando de instalação, algumas dependências e módulos não são instaladas, o que deve ser feito manualmente. Isto resulta numa tarefa desgastante, pois é necessário executar os experimentos e só então com os erros saber o que falta. Nesse trabalho, o *Mininet* foi instalado em uma máquina rodando o *ubuntu* 12.10, devido ao fato da versão disponível para o 12.04 ser muito antiga e com alguns *bugs*. No entanto, a opção de apenas utilizar a VM que o site do emulador disponibiliza não tem esses problemas, basta apenas fazer o *download* e executar com um algum software de virtualização. Ou seja, ela já está completa com todos os ativos e ferramentas que serão necessários para executar experimentos.

A avaliação das métricas de perda de pacotes, latência e largura de banda foi feita seguindo o cenário global uma máquina gateway configura para fazer o roteamento entre uma máquina real e as virtuais emuladas criadas pelo *Mininet*. As máquinas emuladas foram criadas no código em *python* e podiam ser acessadas via *ssh* ou até mesmo abertas no terminal.

O objetivo geral do emulador *Mininet* é permitir experimentos de Redes Definidas por Software (SDN), no entanto como nosso intuito é apenas mostrar que é possível emular e criar experimentos de redes, não utilizamos o *OpenVSwitch* da ferramenta, que define redes virtuais. Assim para criar o cenário definido nas configurações utilizou-se apenas um cenário de roteamento simples. Como não utilizamos *switch* então não teve problema. Mas, no site eles sugerem que quando não for utilizar o *switch openflow*, use o switch para rede legadas, que pode ser feito facilmente com o comando ***sudo mn --switch user --test iperf.***

Depois de configurada, uma máquina host emulada no *Mininet* foi utilizada para comunicar com uma máquina real, seguindo a configuração do cenário global para testar se era possível medir a largura de banda que foi de 94.8 Mbps, a latência da comunicação que foi de 0.060 ms, e a perda de pacotes que foi 102 pacotes, em quantidade. O que demonstra que o *Mininet* pode ser usado para a realização de experimentos que não envolvam apenas SDN, podendo ser configurado para mediar métricas de desempenho de uma rede.

O *Mininet* é um emulador novo mas que está sendo muito utilizado atualmente pela academia em pesquisas de SDN. Assim, pode-se afirmar que ele traz um novo legado para as práticas de redes, não só das redes comuns, mas também no novo paradigma de Redes Definidas por Software.

4 CONSIDERAÇÕES FINAIS

A emulação de Redes permite que facilmente seja criada uma topologia de rede, independente da condição da rede, ou de equipamentos. Assim, utilizar a técnica de emulação de redes no ensino une os conceitos de redes aprendidos teoricamente e que podem ser colocados em práticas com facilidade.

Com os emuladores avaliados pode-se observar que estes estão praticamente no mesmo nível, e os resultados demostraram que é possível coletar métricas em cada emulador. O *Netkit* utilizando de terminais simula bem o funcionamento de redes do mundo real. Apesar de alguns serviços trazerem dificuldades de configurações, é um emulador de fácil manuseio para o usuário, no entanto consome muita memória e usa bastante espaço em disco. Para cada máquina emulada ele ocupa um espaço de 10 a 15 GB no disco, o que dificultaria testes para implementar uma rede grande.

O *Mininet* é o emulador do momento, muito utilizado e não ocupa muito espaço. Já possui muitos serviços implementados, mas exige muitas configurações mesmo para uma criação de rede comum, então seu uso é mais voltado a SDN. Outro ponto é o uso de bibliotecas no *python*, levando um aluno a se preocupar em arquitetar sua topologia e também se preocupar com a sintaxe do código se já não tiver aprendido a linguagem.

O *Core*, apesar da desvantagem de atribuir endereços IPv6 automaticamente a cada *host* quando se cria uma rede, é mais leve e possui uma *GUI* que permite uma maior interação, além de possibilitar ver o funcionamento da rede em tempo real. É mais voltado para testes de serviços no IPv6, talvez esse seja o emulador mais indicado para experimentos de redes em disciplinas de redes. Com muitos serviços implementados, como NAT, IPSec e IPv6, com alguns laboratórios disponíveis utilizando serviços do IPv6, o *Core* passou a ser muito utilizado. Não tem muitos problemas de instalação nem de configuração, com um desempenho notável. Comparando as características positivas e negativas, fica claro que o *Core* é o emulador sugerido desse trabalho.

Finalmente, é importante ressaltar que a escolha de cada emulador dependerá da sua necessidade, portanto quem quiser trabalhar com SDN uma boa escolha seria o *Mininet*; para a virtualização e uma maior robustez o *Netkit* seria o indicado; e para uma criação de experimentos que vão desde o simples ao complexo, com suporte a praticamente quase todos os serviços de redes, o *Core* seria o mais apropriado.

REFERÊNCIAS

AHRENHOLZ, Jeff et al. CORE: A REAL-TIME NETWORK EMULATOR. **Ieee.** Seattle, Wa, p. 1-7. ago. 2008.

CARISSIMI, Alexandre. 2008. Virtualização: da teoria a soluções. **Minicursos do Simpósio Brasileiro de Redes de Computadores** - SBRC´2008, 173-207

COSTA, Giovani Hoff da. **Métricas para Avaliação de Desempenho em Redes QoS sobre IP.** 2008. 42 f. Monografia (Especialização) - Curso de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2008.

FERREIRA, Miguel; BAPTISTA, Ana Alice; RAMALHO, José Carlos. 6. Conferência da Associação Portuguesa de Sistemas de Informação. Bragança, 26-28 Outubro de 2005: Universidade do Ninho, Portugal 2005.

FIOLHAIS, Carlos; TRINDADE, Jorge. Física no Computador: O Computador como uma Ferramenta no Ensino e na Aprendizagem das Ciências Físicas. **Revista Brasileira de Ensino de Física**, São Paulo, v. 25, n. 3, p.259-272, 2003.

GADELHA, Renê N. S. et al. OS Simulator: Um simulador de sistemas de arquivos para apoiar o ensino/aprendizagem de sistemas operacionais. **Anais do XXI Simpósio Brasileiro de Informática na Educação**. João Pessoa, 2010.

GOMES, R. L.; MADEIRA, E. R. M. A Traffic Classification Agent for Virtual Networks Based on QoS Classes. In: IEEE LATIN AMERICA TRANSACTIONS, 3., 2012, Campinas. **Conference.** Campinas: Ieee, 2012. p. 1 – 900.

GURGEL, Paulo Henrique; BARBOSA, Ellen Francine; BRANCO, Kalinka Castelo. A ferramenta Netkit e a virtualização aplicada ao ensino e aprendizagem de redes de computadores. In: XXXII CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (CSBC), 22., 2012, Curitiba. **Anais... .** Curitiba: Congresso da Sociedade Brasileira de Computação, 2012. p. 1 - 10.

GURUPRASAD, S; et al. Integrated network experimentation using simulation and emulation, Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. **Tridentcom 2005**. First International Conference, p.204,212, 23-25 Feb. 2005

HANDIGOL, Nikhil et al. Reproducible Network Experiments Using Container-Based Emulation. **Conext'12**, Nice, France, v. 13, n. 10, p.253-264, jul. 2012.

HERRSCHER, D.; ROTHERMEL, K. A dynamic network scenario emulation tool. Proceedings. **Eleventh International Conference on Computer Communications and Networks**, p.262,267, 14-16 Oct. 2002.

KIDDLE, Cameron. **Scalable Network Emulation.** 2004. 202 f. Tese (Doutorado) - Curso de Computer Science, University Of Calgary, Calgary, 2004.

KROPOTOFF, Alexis Braga. **Um emulador paramétrico de conexões fim-a-fim em redes IP.** 116 f. Dissertação de Mestrado - Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2002.

LAUREANO, Marcos Aurelio Pchek; MAZIERO, Carlos Alberto. Virtualização: Conceitos e Aplicações em Segurança. Minicursos em VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. páginas 139-187. Editora SBC – Porto Alegre, 2008.

LEE, Kyong-ho et al. The State of the Art and Practice in Digital Preservation. **Journal Of Research Of The National Institute Of Standards And Technology,** New York, p. 93-106. fev. 2002.

MARTINS, João Pedro M.L. Sistema para gestão remota de redes experimentais Testbed management systems. Dissertação de mestrado. Portugal. Universidade de Aveiro, 2011.

RIMONDINI, Massimo. **Emulation of Computer Networks with Netkit.** 2007. 42 f. TCC (Graduação) - Curso de Informatica, Università di Roma Tre, Roma, Italy, 2007.

ROTHENBERG, Jeff et al. Avoiding Technological Quicksand: Finding a Viable Technical Foundation for Digital Preservation: A Report to the Council on Library and Information Resources. **Commission On Preservation And Access And Council On Library And Information Resources**, Washington, 1999.

SARI, R.F.; WIRYA, P.L.P., Performance Analysis of Session Initiation Protocol on Emulation Network using NIST NET, **Advanced Communication Technology**, The 9th International Conference on, vol.1, p.506,510, 12-14 Feb. 2007.

TANENBAUM, Andrew S. Redes de Computadores. 4. ed. Rio de Janeiro: Elsevier, 2003.

APÊNDICES

APÊNDICE A – Instalação e Configuração do Emulator Core

- **Instalação**

Máquina Ubuntu Server 12.04 platorma 64 bits.

Primeiro de tudo é necessário atualizar a máquina.

sudo apt-get update

sudo apt-get dist-upgrade

Agora é necessário instalar alguns pacotes, para python e roteamento.

sudo apt-get install bash bridge-utils ebtables iproute libev-dev python tcl8.5 tk8.5 libtk-img

Também é preciso instalar o Quagga para roteamento.

sudo apt-get install quagga

Site para downloads de pacotes do Core.

http://downloads.pf.itd.nrl.navy.mil/core/

Instalação dos pacotes pelo terminal.

sudo dpkg -i core-daemon_4.6-0ubuntu1_precise_amd64.deb

sudo dpkg -i core-gui_4.6-0ubuntu1_precise_all.deb

Iniciando o daemon como root.

sudo /etc/init.d/core-daemon start

Para então executar o Core Gui como usuário normal.

core-gui

- **Ferramentas Utilizadas.**

- Duas máquinas reais e três emuladas
- Ferramenta *Iperf*

- **Configuração**

O core *emulator* conta com uma interface gráfica o que facilita a criação dos cenários. Para criar um cenário basta clicar nas imagens e conectá-los. Para emular um cenário de rede é necessário conectar o *rj-45* ao um roteador. E escolher o protocolo de roteamento.

- **Cenário**

O cenário consiste de 3 máquinas virtuais no core, ligados a roteadores. E a máquina real ligada ao core que funcionará como *gateway* para o outro computador real, configurado na interface eth0.

Na máquina com o emulator core é necessário habilitar o encaminhamento de pacotes na máquina para que ela funcione como um roteador.

echo 1 > /proc/sys/net/ipv4/ip_forward

E então atribuir os Ips e adicionar as rotas para cada máquinas para que esses possam se comunicar.

APÊNDICE B – Instalação e Configuração do Emulator Netkit

- **Instalação**

Como sempre é necessário atualizar as máquinas com o **update e upgrade**. Então partimos para os downloads dos três arquivos essenciais ao *Nekit.*

http://wiki.netkit.org/download/netkit/netkit-2.7.tar.bz2

http://wiki.netkit.org/download/netkit-filesystem/netkit-filesystem-i386-F5.1.tar.bz2

http://wiki.netkit.org/download/netkit-kernel/netkit-kernel-i386-K2.8.tar.bz2

- **Configuração**

Agora depois do Downloader descompacte cada arquivo em um diretório que será ultilizado para os testes.

E então começa a configurar suas variáveis de ambientes. Para que a máquina não perca as configurações a cada inicialização basta editar o arquivo **.bashrc** com os seguintes comandos

export NETKIT_HOME=/home/user/netkit

export MANPATH=:/home/user/netkit/man

export PATH=/home/user/netkit/bin:$PATH

- **Ferramentas**

- Duas máquinas reais e três emuladas.
- Ferramenta iperf

- **Cenário**

Do mesmo modo do core, uma máquina real fez o papel de gateway interligada através da placa eth0. Também é preciso dar o comando de encaminhamento de pacotes no Netkit.

Para criar uma máquina por exemplo, basta usar o comando: Na máquina com o emulador intalado.

vstart myPC1 --mem=100 –eth0=tap,192.168.2.1

No entanto para que a interface de um host virtual se comunique com a de uma placa de rede real é necessário criar uma bridge, semelhante ao que se faz no virtual box e outros sistemas de virtualização.

A criação do link entre as máquinas é feita usando o parâmetro *bridge* ao se declarar a interface de rede. O comado seria:

NomeDaMaquina[eth0]=uplink:bridge=eth0:ip=192.168.2.10/24

Mais antes disso é preciso ter configurado a bridge na máquina real. Para permitir que a virtual emulada se comunique com um pc cliente real.

brctl addbr br0
brctl setfd br0 0
brctl stp br0 off
ifconfig eth0 0.0.0.0 promisc up
brctl addif br0 eth0
dhclient br0
tunctl -u root
ifconfig tap0 0.0.0.0 promisc up
brctl addif br0 tap0

Em seguida reiniciar a sessão UML com o comando:

$NETKIT_HOME/kernel/netkit-kernel ubd0=$NETKIT_HOME/fs/netkit-fs-F3.0a root=98:1 \. mem=100M \. eth0=tuntap,tap0

Com isso já é possível comunicar as máquinas. O programa Quagga responsável pelo roteamento já vem configurado no emulador para ser usado automaticamente. Para criar um roteador utiliza-se o seguinte comando que mostra os hosts conectados a eles.

roteador[type]=router
host1[type]=generic

roteador[ppp0]=link0:ip=10.0.0.1/30
roteador[eth0]=lan0:ip=192.168.0.254/2
roteador[route]=default:dev=ppp0

host1[eth0]=lan0:ip=192.168.0.1/24
host1[default_gateway]=192.168.0.254

APÊNDICE C – Instalação e Configuração do Emulator Mininet

- **Instalação**

Depois de atualizar a máquina é começaremos com a instalação do Mininet.

sudo apt-get install mininet

E outras dependências que ele vai solicitando de acordo com a execução do emulador. Módulo para python.

sudo apt-get install python-setuptools

- **Ferramentas**

– Ferramenta iperf
– Duas Máquinas reais e três emuladas

- **Configuração**

Uma dica é no início deixar as interfaces com as placas para pegar ips automaticamente e então depois configurá-las.

- **Cenário**

Código com um router e três máquinas emuladas... Em Python.

```
"""
Topologia com um router e três máquinas
"""

from mininet.net import Mininet
from mininet.node import Controller, RemoteController
from mininet.log import setLogLevel, info
from mininet.cli import CLI
```

```
from mininet.topo import Topo
from mininet.util import quietRun
from mininet.moduledeps import pathCheck

from sys import exit
import os.path
from subprocess import Popen, STDOUT, PIPE

IPBASE = '10.3.0.0/16'
ROOTIP = '10.3.0.100/16'
IPCONFIG_FILE = '/home/alunotcc/experimento/IP_CONFIG'
IP_SETTING={}

class experimento( Topo ):
    "Topologia do Experimento"

    def __init__( self, *args, **kwargs ):
        Topo.__init__( self, *args, **kwargs )
        host1 = self.add_host( 'host1' )
        host2 = self.add_host( 'host2' )
        router = self.add_switch( 'sw0' )
        root = self.add_host( 'root', inNamespace=False )
        for h in host1, host2, root: #cliente, Usuario root:
            self.add_link( h, router )
```

```
class experimento( Controller ):

   "Experimento multiplos Ips  Bridge"


   def __init__( self, name, inNamespace=False, command='controller',

            cargs='-v ptcp:%d', cdir=None, ip="127.0.0.1",

            port=6633, **params ):

      """
        ip: endereço do controlador
        port: porta do controlador
"""

      Controller.__init__( self, name, ip=ip, port=port, **params)


   def start( self ):

      """Start <controller> <args> on controller.

         Log to /tmp/cN.log"""

      pathCheck( self.command )

      cout = '/tmp/' + self.name + '.log'

      if self.cdir is not None:

         self.cmd( 'cd ' + self.cdir )

      self.cmd( self.command, self.cargs % self.port, '>&', cout, '&' )


   def stop( self ):

      "Stop controller."

      self.cmd( 'kill %' + self.command )

      self.terminate()
```

```
def startsshd( host ):
    "Inicia o sshd no host"
    stopsshd()
    info( '*** Iniciando sshd\n' )
    name, intf, ip = host.name, host.defaultIntf(), host.IP()
    banner = '/tmp/%s.banner' % name
    host.cmd( 'echo "Welcome to %s at %s" >  %s' % ( name, ip, banner ) )
    host.cmd( '/usr/sbin/sshd -o "Banner %s"' % banner, '-o "UseDNS no"' )
    info( '***', host.name, 'executando sshd no', intf, 'at', ip, '\n' )


def stopsshd():

    info( '*** Desligando Processo ',
        quietRun( "pkill -9 -f Banner" ), '\n' )



def starthttp( host ):
    "Inicia um script python "

    info( '*** Iniciando Host 1', host, '\n' )
     #host.cmd( 'cd ~/http_%s/; python -m Host 80 >& /tmp/%s.log &' % (host.name, host.name) )
      #host.cmd( 'cd ~/http_%s/; nohup python2.7 ~/http_%s/webserver.py >& /tmp/%s.log &' % (host.name, host.name, host.name) )
       host.cmd( 'cd ~/http_%s/; nohup python2.7 ~/http_%s/webserver.py &' % (host.name, host.name) )
```

```
            #host.cmd( 'cd ~/http_%s/; screen -S webserver -D -R python2.7 ~/http_
%s/webserver.py ' % (host.name, host.name) )



    def stophttp():
        "para o script python"
        info( '*** Desligando',
            quietRun( "pkill -9 -f SimpleHTTPServer" ), '\n' )
        info( '***Desligando',
            quietRun( "pkill -9 -f webserver.py" ), '\n' )


    def set_default_route(host):
        info('*** Gateway do host %s\n' % host.name)
        if(host.name == 'host1'):
            routerip = IP_SETTING['sw0-eth1']
        elif(host.name == 'host2'):
            routerip = IP_SETTING['sw0-eth2']
        print host.name, routerip
        host.cmd('route add %s/32 dev %s-eth0' % (routerip, host.name))
        host.cmd('route add default gw %s dev %s-eth0' % (routerip, host.name))

        #HARDCODED

        #host.cmd('route del -net 10.3.0.0/16 dev %s-eth0' % host.name)
        ips = IP_SETTING[host.name].split(".")
        host.cmd('route del -net %s.0.0.0/8 dev %s-eth0' % (ips[0], host.name))
```

```
def get_ip_setting():
    if (not os.path.isfile(IPCONFIG_FILE)):
        return -1
    f = open(IPCONFIG_FILE, 'r')
    for line in f:
        if( len(line.split()) == 0):
          break
        name, ip = line.split()
        print name, ip
        IP_SETTING[name] = ip
    return 0


def experimento():
    stophttp()
    "Criando uma simples rede"
    r = get_ip_setting()
    if r == -1:
        exit("não tem endereço ip, check isso %s exists" % IPCONFIG_FILE)
    else:
        info( '*** ip configurado com sucesso para o host \n %s\n' % IP_SETTING)


    topo = experimento()
    info( '*** Criando a Rede\n' )
    #net = Mininet( topo=topo, controller=experimento, ipBase=IPBASE )
```

```
        net = Mininet( topo=topo, controller=experimento, ipBase=IPBASE )
        net.start()
        #host1, host2, router, client = net.get( 'host1', 'host2', 'sw0', 'client')
        host1, host2, router = net.get( 'host1', 'host2', 'sw0')
        s1intf = host1.defaultIntf()
        s1intf.setIP('%s/8' % IP_SETTING['host1'])
        s2intf = host2.defaultIntf()
        s2intf.setIP('%s/8' % IP_SETTING['host2'])

        cmd = ['ifconfig', "eth1"]
        process = Popen(cmd, stdout=PIPE, stderr=STDOUT)
        hwaddr = Popen(["grep", "HWaddr"], stdin=process.stdout, stdout=PIPE)
        eth1_hw = hwaddr.communicate()[0]
            info( '*** setando endereço mac do sw0-eth3 como eth1 (%s)\n' %
eth1_hw.split()[4])
        router.intf('sw0-eth3').setMAC(eth1_hw.split()[4])



        for host in host1, host2:
           set_default_route(host)
        starthttp( host1 )
        starthttp( host2 )
        CLI( net )
        stophttp()
      #   stopsshd()
```

```
    net.stop()


if __name__ == '__main__':
    setLogLevel( 'info' )
    experimento()
```

ANEXOS

ANEXO A – Código básico de uma topologia usando *script python* no *Core.* Esse código está disponível na Documentação Oficial da ferramenta.

```
#!/usr/bin/python

from core import pycore

session = pycore.Session(persistent=True)

node1 = session.addobj(cls=pycore.nodes.CoreNode, name="n1")

node2 = session.addobj(cls=pycore.nodes.CoreNode, name="n2")

hub1 = session.addobj(cls=pycore.nodes.HubNode, name="hub1")

node1.newnetif(hub1, ["10.0.0.1/24"])

node2.newnetif(hub1, ["10.0.0.2/24"])

node1.icmd(["ping", "-c", "5", "10.0.0.2"])

session.shutdown()
```

ANEXO B – Código do *Mininet* para conectar máquinas emuladas com reais. Arquivo hwintf.py, disponível no *Github* da ferramenta na seção de exemplo.

```
#!/usr/bin/python

"""
This example shows how to add an interface (for example a real
hardware interface) to a network after the network is created.
"""

import re, sys

from mininet.cli import CLI
from mininet.log import setLogLevel, info, error
from mininet.net import Mininet
from mininet.link import Intf
from mininet.topolib import TreeTopo
from mininet.util import quietRun

def checkIntf( intf ):
    "Make sure intf exists and is not configured."
    if ( ' %s:' % intf ) not in quietRun( 'ip link show' ):
        error( 'Error:', intf, 'does not exist!\n' )
        exit( 1 )
    ips = re.findall( r'\d+\.\d+\.\d+\.\d+', quietRun( 'ifconfig ' + intf ) )
```

```
    if ips:
        error( 'Error:', intf, 'has an IP address,'
               'and is probably in use!\n' )
        exit( 1 )

if __name__ == '__main__':
    setLogLevel( 'info' )

    # try to get hw intf from the command line; by default, use eth1
    intfName = sys.argv[ 1 ] if len( sys.argv ) > 1 else 'eth1'
    info( '*** Connecting to hw intf: %s' % intfName )

    info( '*** Checking', intfName, '\n' )
    checkIntf( intfName )

    info( '*** Creating network\n' )
    net = Mininet( topo=TreeTopo( depth=1, fanout=2 ) )

    switch = net.switches[ 0 ]
    info( '*** Adding hardware interface', intfName, 'to switch',
          switch.name, '\n' )
    _intf = Intf( intfName, node=switch )

    info( '*** Note: you may need to reconfigure the interfaces for '
          'the Mininet hosts:\n', net.hosts, '\n' )
```

net.start()

CLI(net)

net.stop()

ANEXO C – Laboratório do NAT64 e DNS64 no IPV6, Disponível no Site do Nic.Br

O NAT64 e o DNS64 em conjunto permitem que, em uma rede, *hosts* somente IPv6 acessem servidores somente IPv4 na Internet, por meio da tradução dos protocolos. Nesse laboratório o objetivo é configurar o NAT64 e o DNS64 para verificar o funcionamento da técnica. Primeiramente é preciso instalar o módulo do *kernel* do linux desenvolvido para o projeto *ecdysis*.

1. Para realizar o download, primeiro faça um pequeno cadastro no site do projeto *ecdysis* e, em seguida, você receberá por e-mail o link para baixar os arquivos necessários para instalação. Após a realização do download, descompacte o arquivo e prossiga com a instalação via terminal do seguinte modo:

```
$ tar xvzf ecdysis-nf-nat64-20101117.tar.gz
$ cd ecdysis-nf-nat64-20101117
$ make
$ sudo make install
```

2. Para este experimento, também é necessário o uso do BIND, que é uma implementação do protocolo DNS (Domain Name System), com versão superior a 9.8. Na máquina virtual, utilize um Terminal para rodar os comandos:

```
$ wget ftp://ftp.isc.org/isc/bind9/9.8.1-P1/bind-9.8.1-P1.tar.gz
$ tar xvzf bind-9.8.1-P1.tar.gz
$ cd xvzf bind-9.8.1-P1
$ ./configure
$ make
$ sudo make install
```

3. Inicie o CORE e abra o arquivo “tecnicas-transicao-nat64.imn”, localizada no diretório do desktop “Transição/NAT64”

.

4. Verifique a configuração dos nós da topologia.

Inicie a simulação.

```
# /home/core/simulacao-nat64.sh start
# sudo sysctl -w net.ipv6.conf.all.forwarding=1
```

5. Verifique a conectividade entre cliente e pilhaDupla.

Abra o terminal do cliente através do duplo-clique e utilize o seguinte comando:

```
# ping6 -c 4 2012:2::6:12
```

6. Configure o NAT64 na máquina virtual.

```
# sudo insmod /home/core/ecdysis-nf-nat64-20101117/nf_nat64.ko
nat64_ipv4_addr=192.0.2.2 nat64_prefix_addr=64:ff9b::
nat64_prefix_len=96
# sudo /home/core/nat64-config.sh 192.0.2.2
```

7. Verifique a conectividade via IPv6 entre cliente e v4.

Abra o terminal de cliente através do duplo-clique e utilize o seguinte comando:

```
# ping6 -c 4 64:ff9b::cb00:7102
```

8. Neste passo, verificaremos o funcionamento do serviço DNS sem a opção de DNS64.

a. Abra o terminal da máquina virtual e utilize o seguinte comando para editar configuração DNS: no arquivo **/etc/resolv.conf** e edite para conter apenas 127.0.0.1

c. Abra o terminal da máquina virtual e utilize o seguinte comando para inicializar o serviço DNS:

```
# sudo /usr/local/sbin/named -c /home/core/named.conf
```

d. Abra o terminal de cliente através do duplo-clique e utilize os seguintes comandos:

```
# dig -t ANY servidor.pd
# dig -t ANY servidor.v4
```

9. Neste passo, modificaremos a configuração de DNS para habilitar o DNS64.

a. Abra o terminal da máquina virtual e utilize o seguinte comando para encerrar o serviço: **sudo killall named.**

b. Modifique o arquivo de configuração do DNS através do seguinte comando:

nano /home/core/named.conf e insira o seguinte código:

```
options {
  directory "/home/core/";
  listen-on-v6 { any; };
  dns64 64:ff9b::/96 {
    clients { any; };
  };
};

zone "v4" {
  type master;
  file "v4.zone";
};

zone "pd" {
  type master;
  file "pd.zone";
};
```

Reinicie o serviço DNS .

c. Abra o terminal de cliente através do duplo-clique e utilize os seguintes comandos:

```
# dig -t AAAA servidor.v4
# host servidor.v4
```

O resultado deve ser: