Jonas Diogo Stacke Kercher

Observador didático do funcionamento de protocolos de roteamento para redes IP com interface gráfica

São José – SC

agosto / 2011

Jonas Diogo Stacke Kercher

Observador didático do funcionamento de protocolos de roteamento para redes IP com interface gráfica

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Orientador:
Prof. Evandro Cantú, Dr.

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José – SC

agosto / 2011

Monografia sob o título *"Observador didático do funcionamento de protocolos de roteamento para redes IP com interface gráfica"*, defendida por Jonas Diogo Stacke Kercher e aprovada em 12 de agosto de 2011, em São José, Santa Catarina, pela banca examinadora assim constituída:

Prof. Evandro Cantú, Dr.
Orientador

Prof. Marcelo Maia Sobral, M. Sc.
IFSC

Prof. Emerson Ribeiro de Mello, Dr.
IFSC

Eu acredito demais na sorte. E tenho constatado que, quanto mais duro eu trabalho, mais sorte eu tenho.

Thomas Jefferson

Agradecimentos

Dedico meus sinceros agradecimentos primeiramente àqueles que fizeram que eu chegasse até aqui, meu pai e minha mãe, que me ensinaram que a educação sempre vem em primeiro lugar.

Agradeço ao meu orientador Evandro Cantú, que me apresentou esse trabalho, e a ajuda prestada ao longo de seu desenvolvimento. Agradeço também ao Rodrigo Farias, que possibilitou que eu realizasse esse trabalho, com o seu trabalho já realizado e ajuda prestada.

Agradeço há todos os professores do IFSC da área de telecomunicações, que com os seus conhecimentos passados ao longo desses anos, contribuirão muito para a realização desse trabalho. Agradeço também há todos aos meus colegas de curso e há todos que contribuirão de alguma maneira para que eu pudesse realizar esse trabalho.

Como último agradecimento, agradeço ao CNPq e ao IFSC pela bolsa de pesquisa disponibilizada para a realização do trabalho.

Resumo

O estudo dos protocolos de roteamento na arquitetura TCP/IP pode ser de difícil entendimento se a teoria não vier acompanhada de experiência prática. E a prática não é fácil de ser realizada, porque demanda laboratórios e equipamentos de redes, que nem sempre se tem disponível. Pensando nisso, foi desenvolvido um Observador Didático, onde se possa emular o funcionamento de uma rede completa, em uma única máquina, sem demandar mais equipamentos, e assim visualizar de maneira didática as tabelas de roteamento e as suas modificações ao longo do tempo de forma gráfica.

Abstract

The study of routing protocols in TCP/IP networks can be difficult to understand if the theory is not accompanied by practical experiences. However, practical experiences is not easy to do, because it requires laboratory and networks equipment, which is not always available. To accomplish these tasks, we developed an Didactic Observer of routing protocols, in which we can emulate the operation of a complete network in a single machine, without any more equipment. With the Didactic Observer we can observe the routing tables and their changes over time.

Sumário

Lista de Figuras

1 Introdução p. 13

1.1 Objetivos . . . . . p. 13

1.2 Organização do texto . . . . . p. 14

2 Fundamentação teórica e estado atual do Observador Didático p. 15

2.1 Protocolos de roteamento . . . . . p. 15

2.1.1 Algoritmos Estado de Enlace . . . . . p. 16

2.1.2 Algoritmos Vetor de Distâncias . . . . . p. 16

2.1.3 Protocolo de Roteamento RIP . . . . . p. 17

2.1.4 Protocolo de Roteamento OSPF . . . . . p. 17

2.2 Observador Didático . . . . . p. 18

2.3 Máquinas Virtuais UML . . . . . p. 19

2.3.1 Inicializando uma máquina UML . . . . . p. 20

2.3.2 Mecanismo *Copy-On-Write* (COW) . . . . . p. 21

2.3.3 Criando redes . . . . . p. 23

2.3.4 Iniciando uma Rede Complexa . . . . . p. 25

2.4 Protocolo SNMP . . . . . p. 25

2.5 Funcionamento do Observador Didático e seus aspectos construtivos . . . . . p. 29

2.5.1 Resultados obtidos e considerações . . . . . p. 32

2.5.2 Melhorias pretendidas . . . . . p. 35

2.6 Netkit . . . . . p. 35

2.6.1 Usando Netkit . . . . . p. 36

2.7 Graphviz . . . . . p. 41

3 Implementação das melhorias p. 43

3.1 Implementação do Observador Didático utilizando o Net-kit . . . . . p. 43

3.2 Problema encontrado com o Net-SNMP . . . . . p. 47

3.3 Implementação de uma saída gráfica para o Observador Didático . . . . . p. 49

3.3.1 Implementação de um programa para gerar a saída gráfica . . . . . p. 52

3.4 Extensão da ferramenta para observação de outros protocolos de roteamento . p. 54

4 Resultados e testes p. 55

4.1 Executando um Laboratório com o Observador Didático . . . . . p. 55

4.2 Executando um Laboratório com o Observador Didático utilizando outro protocolo de roteamento . . . . . p. 62

4.3 Limitações encontradas com a implementação realizada . . . . . p. 64

5 Conclusões p. 66

Anexo A – Exemplo de uso das máquinas virtuais UML p. 67

Anexo B – Preparando a ferramenta Netkit para ser usada e exemplo de uso p. 69

B.1 Preparando Netkit para ser usado . . . . . p. 69

B.2 Exemplo de uso da ferramenta Netkit . . . . . p. 70

Anexo C – Usando a ferramenta Graphviz e exemplo de uso p. 73

C.1 Usando Graphviz . . . . . p. 73

C.2 Exemplo de uso da ferramenta Graphviz . . . . . p. 74

Anexo D – Preparar Laboratório do Netkit para ser usado com o Observador Didático p. 77

Lista de Abreviaturas p. 84

Referências Bibliográficas p. 85

Lista de Figuras

2.1 Relação entre UML e o hospedeiro (DIKE, 2006). . . . . p. 20

2.2 Opção para informar um arquivo como um dispositivo raiz. . . . . p. 21

2.3 Outras opções para configurar as máquinas virtuais. . . . . p. 21

2.4 Comando mais completo para a inicialização de uma máquina virtual UML. . p. 21

2.5 Mecanismo COW. . . . . p. 22

2.6 Opção para iniciar uma máquina virtual utilizando mecanismo COW. . . . . p. 23

2.7 Criando uma interface TUN/TAP e conectando-a há máquina virtual. . . . . p. 23

2.8 Removendo as interfaces TUN/TAP criadas. . . . . p. 24

2.9 Criando uma rede utilizando UML Switch. . . . . p. 24

2.10 Inicializando uma máquina virtual UML utilizando Xterm. . . . . p. 25

2.11 Árvore de identificadores de objetos ASN.1 (KUROSE; ROSS, 2006). . . . . p. 27

2.12 Cenário de rede do Observador Didático (FARIAS, 2010). . . . . p. 30

2.13 Observador Didático em funcionamento (FARIAS, 2010). . . . . p. 31

2.14 Mudanças nas tabelas de roteamento com o Observador Didático em execução (FARIAS, 2010). . . . . p. 33

2.15 Mudança na tabela de roteamento apos o enlace Brasil-Itália ser derrubado (FARIAS, 2010). . . . . p. 34

2.16 *Script* para executar um laboratório do Netkit utilizando os V comandos. . . . p. 38

2.17 Exemplo de arquivo `lab.conf` e a topologia gerada (BATTISTA et al., 2007). p. 39

2.18 Um grafo com 6 vértices e 7 arestas. . . . . p. 41

3.1 Arquivo `lab.conf` criado. . . . . p. 44

3.2 Arquivo `lab.dep` criado. . . . . p. 45

3.3 Estrutura do Observador Didático utilizando o Netkit. . . . . p. 46

3.4 Observador Didático utilizando o Netkit em funcionamento. . . . . p. 46

3.5 MIB do Net-SNMP contendo configurações dos agentes. . . . . p. 48

3.6 Imagem da topologia de rede e tabelas de roteamento gerada com a ferramenta Graphviz. . . . . p. 50

3.7 Imagem gerada com a ferramenta Graphviz representando três nós de rede em um mesmo domínio de colisão. . . . . p. 51

3.8 Observador Didático em funcionamento com a visualização em forma gráfica. p. 54

4.1 Tabela de roteamento capturada - 01 - RIP . . . . . p. 56

4.2 Tabela de roteamento capturada - 03 - RIP . . . . . p. 57

4.3 Tabela de roteamento capturada - 04 - RIP . . . . . p. 57

4.4 Tabela de roteamento capturada - 06 - RIP . . . . . p. 58

4.5 Tabela de roteamento capturada - 07 - RIP . . . . . p. 59

4.6 Tabela de roteamento capturada - 11 - RIP . . . . . p. 60

4.7 Tabela de roteamento capturada - 12 - RIP . . . . . p. 60

4.8 Tabela de roteamento capturada - 13 - RIP . . . . . p. 61

4.9 Tabela de roteamento capturada - 06 - OSPF . . . . . p. 62

4.10 Tabela de roteamento capturada - 07 - OSPF . . . . . p. 63

4.11 Tabela de roteamento capturada - 09 - OSPF . . . . . p. 64

A.1 Topologia de rede usada para o exemplo. . . . . p. 67

A.2 *Script* que inicializa a rede utilizada no exemplo. . . . . p. 68

B.1 Configurando as variáveis de ambiente para executar os comandos do Netkit. p. 69

B.2 Topologia de rede usada para o exemplo. . . . . p. 70

B.3 Arquivo `lab.conf` da rede utilizada no exemplo. . . . . p. 71

B.4 Diretório do laboratório do Netkit. . . . . p. 71

B.5 Exemplo dos arquivos `.startup` da rede utilizada no exemplo. . . . . p. 72

C.1 Exemplo da descrição de um grafo simples. . . . . p. 73

C.2 Imagem gerada com o programa `circo` da ferramenta Graphviz. . . . . p. 74

C.3 Arquivo que descreve o grafo para o fluxograma do exemplo. . . . . p. 75

C.4 Fluxograma gerado com o programa `dot` da ferramenta Graphviz. . . . . p. 76

D.1 Topologia de rede usada para o exemplo. . . . . p. 77

D.2 Arquivo `lab.conf` da rede da Figura D.1. . . . . p. 78

D.3 Alterações no arquivo `lab.conf`. . . . . p. 78

D.4 Estrutura do laboratório após as modificações. . . . . p. 79

D.5 Exemplo de um arquivo `.startup`. . . . . p. 79

D.6 Exemplo de um arquivo `.startup` de uma máquina virtual que é um roteador. p. 80

D.7 Criando diretórios e arquivos necessários. . . . . p. 81

D.8 Criando diretórios e arquivos necessários. . . . . p. 81

D.9 Modificações necessárias no arquivo `snmpd.conf`. . . . . p. 81

D.10 Exemplo do `script` `inFile.sh`. . . . . p. 83

D.11 Configuração do arquivo `lab.dep`. . . . . p. 83

1 Introdução

O estudo e aprendizagem dos protocolos e algoritmos de roteamento é um conhecimento indispensável para os profissionais da área de redes de computadores e telecomunicações. O aprendizado destes conhecimentos é dificultado se não houver exemplos práticos onde seja possível visualizar os conceitos aprendidos na parte teórica. A realização de atividades práticas nem sempre é de fácil execução, uma vez que demanda equipamentos de rede, os quais nem sempre se tem disponível, além de problemas e dificuldades para colocar em funcionamento a estrutura física necessária para a realização da prática.

Pensando nisso Farias e Cantú (2010) propuseram e desenvolveram um Observador Didático do funcionamento dos protocolos de roteamento, onde é possível visualizar a montagem das tabelas de roteamento nos roteadores e suas alterações quanto um enlace é derrubado. A proposta inicial era que o Observador Didático pudesse ser utilizado para ensinar os algoritmos e protocolos de roteamento, para isso as tabelas de roteamento deveriam ser exibidas de forma gráfica e o Observador Didático fosse de fácil utilização para observar qualquer cenário possível. Mas esse objetivo não foi totalmente alcançado, a montagem das tabelas de roteamento não são exibidas de forma gráfica, sendo visualizadas em modo texto e o Observador Didático desenvolvido observa as mudanças sempre numa mesma topologia de rede e usando o mesmo protocolo de roteamento.

1.1 Objetivos

Esse trabalho visa dar continuidade ao trabalho já iniciado por Farias (2010), com objetivo de aprimorar o Observador Didático, implementando várias melhorias e com isso ser usado no ensino dos algoritmos e protocolos de roteamento.

Para esse objetivo ser atingido ele pode ser dividido em objetivos menores, que realizados terão conseguido aprimorar o Observador Didático e assim feito várias melhorias no mesmo. Esses objetivos são:

- Aperfeiçoamento do método para construir os cenários de redes, utilizando a ferramenta Netkit. Com isto também pretende-se generalizar a estrutura física atualmente utilizada como cenário de observação para outros cenários;
- Implementação de uma saída gráfica para o Observador Didático, utilizando a ferramenta Graphviz, que é uma linguagem em formato texto, que pode ser compilada gerando figuras gráficas;
- Extensão da ferramenta para observação de outros protocolos de roteamento para poder entender seu funcionamento e comportamento.

1.2 Organização do texto

O texto está organizado da seguinte forma: No capítulo 2 é apresentado uma breve explicação sobre os protocolos de roteamento, o Observador Didático já construído e seu funcionando juntamente com seus resultados. Em seguida são discutidas as melhorias pretendidas, e são explicadas as ferramentes e tecnologias usadas. No capítulo 3 é apresentado como a implementação das melhorias foram feitas no Observador Didático. No capítulo 4 são apresentados os testes e resultados obtidos com o Observador Didático depois de implementadas as melhorias. Por fim, no capítulo 5 são apresentadas as conclusões sobre este trabalho.

2 *Fundamentação teórica e estado atual do Observador Didático*

Neste capítulo será descrito o Observador Didático realizado por Farias (2010), como foi feita sua implementação e os resultados que foram obtidos. Depois serão mostradas as melhorias pretendidas para poder se aprimorar o Observador Didático. Durante toda essa descrição, quando necessário, serão abordados os temas e tecnologias que necessitam de uma explicação para o entendimento do trabalho realizado. Mas primeiramente será feito uma breve explicação dos protocolos de roteamento.

2.1 Protocolos de roteamento

O roteamento é uma das tarefas mais importantes nas redes IP pois permite determinar a rota pela qual um pacote contendo uma determinada informação deve ser encaminhado.

A atividade de roteamento envolve duas tarefas: o roteamento e o encaminhamento de pacotes. O roteamento consiste na definição de rotas para transmitir os pacotes. O encaminhamento é a comutação dos pacotes que chegam aos enlaces de entrada dos roteadores para os enlaces de saída apropriados, fazendo uso de tabelas de roteamento (KUROSE; ROSS, 2006).

O processo de roteamento é o responsável pela montagem e manutenção das tabelas de roteamento nos roteadores. Este processo é executado por um protocolo de roteamento, o qual pode ser dinâmico ou estático. O roteamento estático representa as ações do administrador de rede para alterar as tabelas de roteamento manualmente e o roteamento dinâmico representa as ações de algum protocolo de roteamento sobre as tabelas de roteamento. O roteamento dinâmico é baseado em algoritmos que sempre estão atentos a mudanças na rede e podem ficar alterando as tabelas de roteamento de acordo com a necessidade.

Os algoritmos de roteamento dinâmicos mais conhecidos podem ser classificados em algoritmos globais ou descentralizados. Os algoritmos globais têm o conhecimento completo e global da rede fazendo com que seus cálculos sejam melhores, porém necessitam de todas as

informações necessárias sobre a rede antes de começar os cálculos. Os algoritmos descentralizados fazem o cálculo das rotas de maneira iterativa e distribuída, ou seja, o cálculo da melhor rota é realizado necessitando apenas ter o conhecimento necessário dos seus enlaces e da troca de informações com seus roteadores vizinhos (KUROSE; ROSS, 2006).

Os dois principais algoritmos de roteamento são o algoritmo Estado de Enlace, um algoritmo global, e o algoritmo Vetor de Distâncias, um algoritmo descentralizado.

2.1.1 Algoritmos Estado de Enlace

Os algoritmos estado de enlace são algoritmos globais, sendo assim eles necessitam saber os custos de todos os enlaces entre roteadores de toda a rede. Para que todos os roteadores da rede obtenham tais informações cada roteador deve executar duas importantes tarefas. Primeiro, verificar o estado dos seus enlaces periodicamente, para tentar detectar qualquer mudança no estado do enlace que implique em alterações de custo para ele. Segundo, propagar informações periodicamente do estado de seus enlaces para todos os outros roteadores (COMER, 2006) (FARIAS, 2010).

A tabela de roteamento de cada roteador é atualizada sempre que chega uma nova informação diferente da atual em sua tabela. O algoritmo Estado de Enlace também é conhecido como algoritmo de Dijkstra. Este algoritmo, dispondo de todos os custos de todas as rotas da rede, calcula a melhor rota para cada um dos n roteadores destino da rede em n iterações (KUROSE; ROSS, 2006). Com isso, define-se em cada uma das iterações a melhor rota para um possível roteador de destino (FARIAS, 2010). Um protocolo de roteamento que implementa o algoritmo Estado de Enlace é o protocolo de roteamento *Open Shortest Path First* (OSPF).

2.1.2 Algoritmos Vetor de Distâncias

Os algoritmos vetor de distâncias, também conhecido como algoritmo de Bellman- Ford, ao contrário dos algoritmos estado de enlace (que são globais), são algoritmos distribuídos. O roteador que faz uso do algoritmo vetor de distâncias necessita apenas trocar mensagens entre seus vizinhos, pois necessita somente da informação contida neles para executar seus cálculos e transmiti-los (KUROSE; ROSS, 2006) (FARIAS, 2010).

Ao contrário do algoritmo estado de enlace que precisa saber de todos os custos da rede para executar seus cálculos, o roteador que faz uso do algoritmo vetor de distâncias necessita apenas dos custos de seus enlaces para dar início aos seus cálculos de roteamento. Conforme seus vizinhos vão recebendo novas informações, processando-as e disseminando- as entre seus

vizinhos, este roteador, anteriormente mencionado, vai descobrindo novos destinos (novas redes), calculando melhores caminhos e disseminando suas novas descobertas. Este processo continua até que não haja novas alterações nas tabelas de roteamento (KUROSE; ROSS, 2006) (FARIAS, 2010). Um protocolo de roteamento que implementa o algoritmo Vetor de Distâncias é o protocolo de roteamento *Routing Information Protocol* (RIP).

2.1.3 Protocolo de Roteamento RIP

O protocolo RIP estabelece um custo unitário para cada enlace fazendo assim com que seja um protocolo que conta saltos para calcular a melhor rota para um destino. Sendo assim, só é necessário recalcular as rotas quando um enlace cai ou um roteador apresenta problemas, pois o custo dos enlaces nunca irá mudar (STEVENS, 2008) (FARIAS, 2010).

O RIP define como métrica máxima 15 saltos. Esta definição limita o tamanho da rede onde este protocolo está sendo executado. Um roteador, para divulgar o seu status para seus roteadores vizinhos diretamente conectados a si, envia mensagens periodicamente a cada 30 segundos contendo informações sobre suas rotas. Se tais mensagens de um determinado roteador não são recebidas por seus vizinhos dentro de 3 minutos, estes definem na tabela de roteamento a métrica infinita (16) para ele. Um roteador após definir a métrica infinita para um outro roteador vizinho, aguarda 60 segundos para apagar a rota da tabela de roteamento (STEVENS, 2008) (FARIAS, 2010).

As mensagens RIP são transportadas através do protocolo UDP e destinadas à porta 520. Cada mensagem de atualização periódica pode comportar informações da tabela de roteamento de até 25 rotas (STEVENS, 2008) (FARIAS, 2010).

2.1.4 Protocolo de Roteamento OSPF

O protocolo OSPFv2 é definido no RFC 2178 e como foi concebido posteriormente ao RIP, ele é um protocolo com uma série de características avançadas. Um roteador que executa este protocolo envia o estado de seus enlaces periodicamente a todos os roteadores da rede (KUROSE; ROSS, 2006) (FARIAS, 2010).

Todas as mensagens trocadas entre roteadores OSPF são certificadas. O OSPF permite que um destino possa ter mais de um caminho no caso de custo igual, possibilitando a distribuição da carga entre os diferentes caminhos (KUROSE; ROSS, 2006) (FARIAS, 2010).

O OSPF permite diferentes tipos de métricas para diferentes tipos de tráfegos, ou seja,

pacotes com ToS diferente com um destino comum podem ser enviados por rotas diferentes. Assim o OSPF pode definir caminhos com métricas baixas para mensagens que necessitam de altas taxas de transmissão e caminhos com métricas mais elevadas para mensagens que não necessitam de altas taxas de transmissão. O OSPF também tem suporte integrado para roteamento unicast e multicast (KUROSE; ROSS, 2006) (FARIAS, 2010).

Um dos maiores avanços do OSPF foi possibilitar a criação de uma hierarquia dentro de uma rede resolvendo problemas de escalabilidade. Isso é possível através da criação de áreas dentro da rede onde cada área executa seu próprio algoritmo estado de enlace independentemente uma da outra. Cada área deve conter um ou mais roteadores de borda responsáveis por interagir com outros roteadores de borda de outras áreas. Tais roteadores também podem se comunicar com outros roteadores, que não pertencem a nenhuma área, conhecidos como roteadores de *backbone*, pois estão um nível acima dos roteadores inclusos em áreas. Interconectado com os roteadores de *backbone* deve estar um roteador de borda da rede responsável por trocas informações de roteamento com outros roteadores de borda de outras redes (KUROSE; ROSS, 2006) (FARIAS, 2010).

2.2 Observador Didático

O estudo e aprendizado do funcionamento dos protocolos de roteamento pode ser facilitado observando-se o funcionamento de roteadores em operação numa rede real. Mas isso requer ter disponíveis vários equipamentos e de toda uma estrutura para poder-se observar os protocolos de roteamento, além de demandar um trabalho extra para a preparação de toda rede, como montagem dos equipamentos, cabeamento e a conexão lógica entre eles. Ao invés de usar uma rede real, pode ser utilizado máquinas virtuais para fazer emulação do que se quer observar. Existem disponíveis inúmeros softwares para a criação de máquinas virtuais, como Virtual Box e VMware, mas que usam muito recursos da máquina hospedeira, inviabilizando a criação de várias máquinas virtuais para se fazer um experimento maior. Por isso optou-se pelo uso de máquinas virtuais *User-Mode Linux* (UML) para a construção do Observador Didático pela sua simplicidade, por serem de um projeto de código livre e por ser desenvolvido para o sistema operacional Linux (FARIAS, 2010).

Disponibilizando-se de uma rede composta de máquinas virtuais para rodar os protocolos de roteamento, pode-se utilizar um protocolo de gerenciamento de redes, como o *Simple Network Management Protocol* (SNMP), para se fazer o acompanhamento e monitoramento do funcionamento de tais protocolos. Essa ideia foi utilizada para a construção do Observador

Didático, utilizando máquinas virtuais UML e usando o protocolo SNMP para a observação do funcionamento dos protocolos de roteamento. A seguir serão explicados as máquinas virtuais UML e o protocolo SNMP para depois descrever o funcionamento do Observador Didático proposto.

2.3 Máquinas Virtuais UML

Uma máquina virtual UML é uma máquina Linux que roda em um sistema operacional Linux. Tecnicamente, a UML é uma implementação de um sistema Linux rodando sobre outro sistema Linux. Linux foi implementado para vários processadores, incluindo o x86, Sun SPARC, IBM e Motorola PowerPC, DEC Alpha (então, Compaq e HP), e uma variedade de outros. UML é uma implementação do Linux exatamente no mesmo sentido que os outros. A diferença é que ela é uma implementação para a interface de software definido pelo Linux ao invés da interface de hardware definida pelo processador e do resto do computador físico (DIKE, 2006).

As máquinas virtuais UML são máquinas Linux completas rodando em um computador hospedeiro Linux. Elas tem toda a parte lógica igual a qualquer outra máquina Linux, rodando todos os softwares e serviços como qualquer outra máquina real. O diferencial do UML em relação há outros *softwares* de criação de máquinas virtuais é que pode ser executado um maior número de máquinas virtuais ao mesmo tempo, compartilhando o mesmo *hardware* do hospedeiro, as quais podem ser descartadas quanto não são mais necessárias (DIKE, 2006). O funcionamento dos protocolos de roteamento nas máquinas virtuais UML será igual ao de uma máquina real, podendo ser observado o seu real funcionamento.

Algo que tem que ficar bem definido é a relação entre uma máquina UML e o computador hospedeiro. O UML roda sobre o hospedeiro e não pode existir sem ele, no entanto em termos lógicos, são totalmente separados um do outro. Um usuário pode ter privilégios de super usuário (*root*) dentro da máquina UML, mas não ter os mesmos privilégios sobre o hospedeiro.

Essa relação entre o UML e o hospedeiro pode ser melhor entendida olhando a Figura 2.1. A UML é ao mesmo tempo, um *kernel* (componente central do sistema operacional) Linux para a máquina virtual e um processo Linux para o hospedeiro. Para o *kernel* do hospedeiro, a UML é considerado um processo normal, já para os processo da UML, ela é considerada um *kernel*. Os processos podem interagir com o *kernel* através de chamadas de sistema (*system calls*), que são procedimentos para o *kernel* fazer algo em seu nome. Então UML faz chamadas de sistema para o hospedeiro, o que o torna um processo, e implementa as chamadas de sistema para os

seus próprios processos, tornando-se um *kernel* (DIKE, 2006).

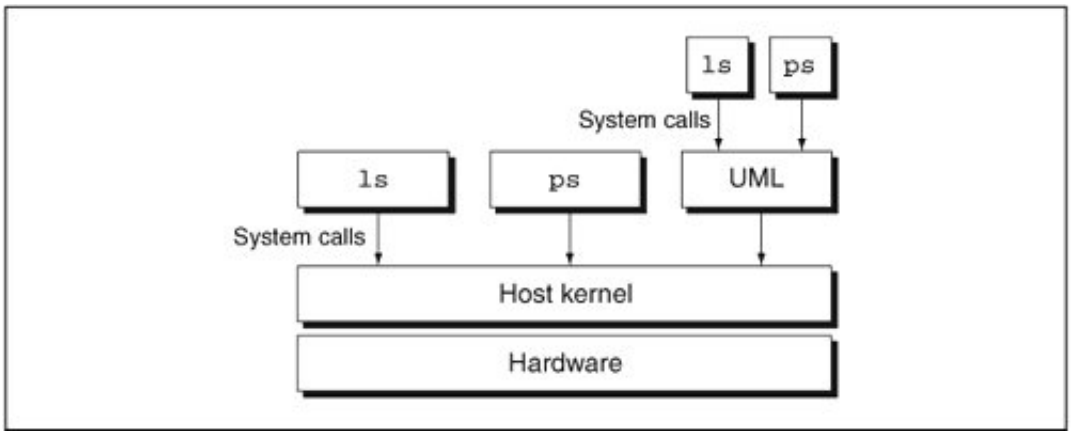


Figura 2.1: Relação entre UML e o hospedeiro (DIKE, 2006).

2.3.1 Inicializando uma máquina UML

Para iniciar uma máquina UML precisa-se ter disponível um aplicativo que quanto executado será o *kernel* da máquina virtual, simulando o hardware virtual e interagindo com o computador hospedeiro como um processo comum. Essa ferramenta é conhecida como UML Kernel. Executando essa ferramenta em um terminal de linha de comando sem parâmetros nenhum, o resultado obtido será muito parecido com o que acontece no *boot* (inicialização) de uma máquina Linux normal. Observando esses resultados pode-se constatar que o *boot* não será bem sucedido, e várias mensagens aparecerão, como por exemplo, indicando que os subsistemas do sistema de arquivos e redes são iguais ao *boot* de um Linux normal. O restante das mensagens embora diferentes tem o mesmo propósito, devido principalmente ao fato que os *drivers* de inicialização do *hardware* da UML serem diferentes do hospedeiro, já que UML não tem o mesmo hardware e uma arquitetura de *kernel* diferente do hospedeiro.

O motivo para que o *boot* da máquina virtual tenha falhado ao executar-se o UML Kernel é que não foi fornecido um dispositivo raiz, de modo que a máquina virtual não pode montar seu sistema de arquivos. A primeira coisa a fazer é dar a UML um dispositivo raiz adequado para que ela tenha condições de inicializar. Os dispositivos UML são virtuais e são construídos a partir de recursos do hospedeiro, então os discos UML são geralmente arquivos no sistema de arquivos do hospedeiro que simulam um disco rígido para o UML Kernel onde ficam gravados todos os arquivos de programas e de usuários da máquina virtual.

Então a UML irá ler e escrever sobre este arquivo no hospedeiro que simula um disco rígido, como se fosse feito em um disco físico. Para fornecer esse arquivo como um dispositivo raiz,

é preciso informar a UML para juntar-se a esse arquivo (DIKE, 2006). Isso é feito através da opção mostrada na Figura 2.2 no terminal da linha de comando.

```
ubda=<sistema de arquivos>
```

Figura 2.2: Opção para informar um arquivo como um dispositivo raiz.

Outras opções que podem ser acrescentadas para fazer configuração da máquina virtual são mostradas na Figura 2.3.

```
umid=<nome maquina> mem=128M
```

Figura 2.3: Outras opções para configurar as máquinas virtuais.

A primeira opção da Figura 2.3 define um nome para a máquina virtual, assim ela será conhecida por esse nome. Se não for usada essa configuração, a UML irá ganhar um nome aleatório. A segunda opção define para a UML 128 MB de memoria física, mas isso não ocupa realmente 128 MB da memória no hospedeiro. Pelo contrario, será criado um arquivo esparso (ocupa espaço à medida que se for escrevendo nele) de 128 MB no hospedeiro. Sendo esparso, esse arquivo vai ocupar muito pouco espaço até que os dados comecem a ser escritos neles. Sendo esse arquivo de tamanho fixo, a UML ficará limitada a essa quantidade de memória. Desde o *boot* a alocação de memória é feita dinamicamente, conforme necessário para a máquina UML, fazendo com que o consumo real seja menor ou igual ao máximo. Conservando assim a memória do hospedeiro e tornando possível a execução de um número bem maior de máquinas virtuais (DIKE, 2006).

Um comando mais completo para a inicialização de uma máquina UML pode ser vista na Figura 2.4, podendo escolher a quantidade de memória que se queira e o nome da máquina.

```
./<UMLKernel> umid=<nome maquina> mem=128M ubda=<sistema de arquivos>
```

Figura 2.4: Comando mais completo para a inicialização de uma máquina virtual UML.

2.3.2 Mecanismo *Copy-On-Write* (COW)

Vimos como iniciar uma máquina UML, mas somente uma máquina quase sempre não atende nossas necessidades para fazer experimentos. Precisamos executar mais de uma máquina virtual. Cada máquina virtual precisa de um arquivo para ser o seu sistema de arquivos, e esses

arquivos geralmente não são pequenos. Utilizar um arquivo separado para cada UML se torna inviável. Entretanto se inicializarmos duas máquinas usando o mesmo arquivo para o sistema de arquivos teremos problemas, se ambas ficarem modificando o mesmo arquivo, uma ficará sobrescrevendo as mudanças da outra (na verdade isso nem chega acontecer porque ocorrerá um erro antes).

Para contornar esse problema é utilizado um mecanismo chamado *Copy-On-Write* (COW), que permite que várias máquinas UML compartilhem um único arquivo no hospedeiro para o sistema de arquivos. Uma COW consiste em um arquivo secundário para cada UML e nele ficarão gravados todas as mudanças feitas no sistema de arquivos, mantendo o sistema de arquivo inalterável a cada utilização. O seu funcionamento pode ser explicado pela Figura 2.5. Inicialmente o arquivo COW começa sem blocos válidos e o arquivo do sistema de arquivos totalmente preenchido. Se um processo lê um bloco, irá obter os dados que estão no arquivo do sistema de arquivos. Se esse bloco é gravado de volta, os novos dados serão gravados para o bloco corresponde no arquivo COW, a partir daí, o bloco original nunca mais será lido e todas as leituras subsequentes serão feitas do arquivo COW (DIKE, 2006).

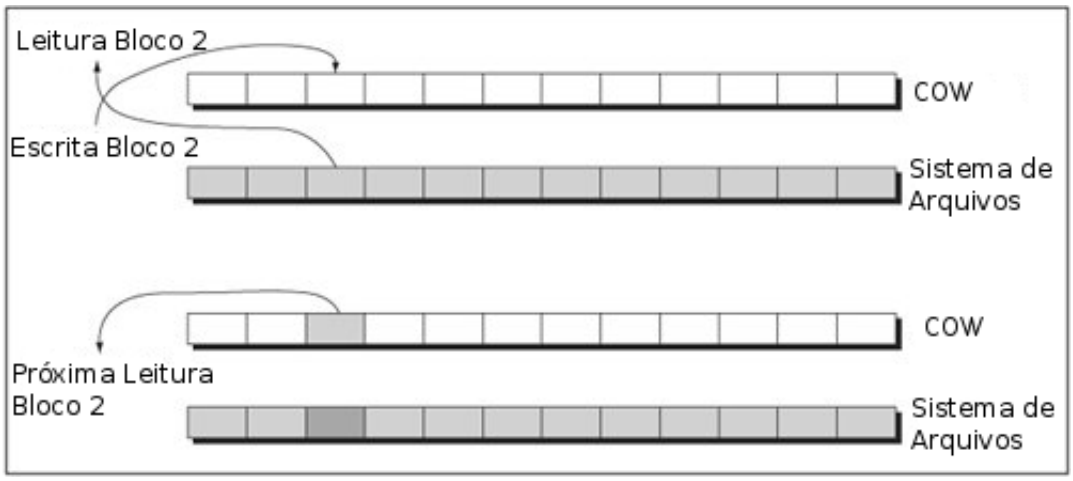


Figura 2.5: Mecanismo COW.

Os arquivos COW também são esparsos e só ocuparão espaço há medida que o sistema de arquivo for sendo alterado, podendo chegar ao mesmo tamanho do arquivo do sistema de arquivos se todos os blocos forem modificados. Para iniciar uma máquina UML usando COW, deve se passar um parâmetro a mais na opção que define o sistema de arquivos, separadas por virgula, o primeira parâmetro será o arquivo COW e a segundo o sistema de arquivos, como pode ser visto na Figura 2.6.

```
ubda=<COW>,<sistema de arquivos>
```

Figura 2.6: Opção para iniciar uma máquina virtual utilizando mecanismo COW.

2.3.3 Criando redes

Agora que já podemos inicializar várias máquinas virtuais UML, precisamos saber como conectar umas as outras, criando as redes que precisamos para fazer os experimentos. Existem algumas maneiras de se fazer essas conexões, iremos mostrar apenas aquelas que foram usadas no Observador Didático, que são duas. Uma delas faz as conexões entre as máquinas virtuais e a outra conecta-as ao hospedeiro, podendo servi-las com conexão à internet.

Para se fazer a conexão entre a máquina virtual e o hospedeiro precisa-se criar e configurar entre as duas uma interface virtual TUN/TAP, que oferece a transmissão e recepção de pacotes entre os programas do usuário. Podendo ser visto como um dispositivo ponto-a-ponto ou uma simples interface *Ethernet*, que em vez de receber pacotes do meio físico, vai recebê-los de um programa do usuário e enviá-los para outro programa. Sendo TAP um dispositivo de rede virtual *Ethernet*, trabalhando com frames *Ethernet* e o TUN sendo o dispositivo de rede virtual ponto-a-aponto trabalhando com datagramas IP (KRASNYANSKY, 2000).

Primeiramente deve-se criar no hospedeiro a interface virtual TUN/TAP, utilizando um aplicativo que desempenha esse papel, o tuntcl, depois deve-se ativar essa interface e configurá-la com um endereço IP. Com isso a interface estará pronta para ser usada, só precisando ser configurada na máquina virtual UML para ela se conectar a essa interface. Isso é feito na inicialização da máquina UML passando um parâmetro a mais que pode ser observado na Figura 2.7, juntamente com a criação da interface TUN/TAP.

```
./tuntcl -U <nome do usuario>
ip link set dev tap0 up
ip address add X.X.X.X/X dev tap0
./<UMLKernel> umid=<nome maquina> mem=128M ubda=<COW>,<sistema de arquivos> ethX
    =tuntap,tap0>
```

Figura 2.7: Criando uma interface TUN/TAP e conectando-a há máquina virtual.

Esse parâmetro consiste em passar à interface da máquina virtual que será conectada, seguido do tipo da conexão, nesse caso tuntap, e separado por vírgula, a interface a qual será conectada, que é a interface criada no hospedeiro. É necessário ser usuário *root* no hospedeiro para a criação das interfaces TUN/TAP, e se for criado mais de uma interface TUN/TAP elas receberam nomes sequenciais tap0, tap1, etc. Depois de realizado o experimento as interfaces

TUN/TAP precisam ser derrubadas e removidas do hospedeiro como mostrado na Figura 2.8.

```
ip link set dev tap0 down
./tunctl -d tap0
```

Figura 2.8: Removendo as interfaces TUN/TAP criadas.

Para realizar as conexões entre as máquinas virtuais pode ser utilizado um aplicativo que simula um *switch* virtual, sendo conhecido como UML Switch, podendo-se ligar as máquinas virtuais a ele. UML Switch é um processo que implementa um *switch* virtual, onde as máquinas virtuais se conectam com ele através de um soquete de domínio UNIX no hospedeiro. Ele pode atuar como um *switch*, sua operação normal e também como um *hub*, o que é útil quando se precisa capturar o tráfego entre duas máquinas virtuais a partir de uma terceira. O *switch* também tem a capacidade de se conectar a uma interface TUN/TAP já configurada, permitindo que as máquinas virtuais ligadas a ele posam se comunicar com o hospedeiro e a rede externa (DIKE, 2006).

Para cada rede que se precisa construir é lançado um UML Switch. Depois conecta-se a cada *switch* as máquinas virtuais com as suas interfaces correspondentes para se criar as redes. Primeiro deve-se criar todos os *switches* necessários utilizando o UML Switch, passando como parâmetro a criação de um arquivo no hospedeiro que representará o soquete UNIX criado. Depois deve-se iniciar as máquinas virtuais utilizando o mesmo parâmetro utilizado para a conexão nas interfaces TUN/TAP, trocando o tipo para *daemon* e depois separados por três vírgulas, o arquivo que representa o soquete há qual deve se conectar, como pode ser visto na Figura 2.9.

```
./uml_switch -unix /tmp/switch1.ctl &
./<UMLKernel> umid=<nome maquina1> mem=128M ubda=<COW1>,<sistema de arquivos>
    ethX=daemon,,,/tmp/switch1.ctl &
./<UMLKernel> umid=<nome maquina2> mem=128M ubda=<COW2>,<sistema de arquivos>
    ethX=daemon,,,/tmp/switch1.ctl &
```

Figura 2.9: Criando uma rede utilizando UML Switch.

No exemplo da Figura 2.9 as duas máquinas virtuais estão conectadas ao mesmo *switch* virtual, e assim, estando na mesma rede, as duas podem se comunicar. As vírgulas seguidas sem nada entre elas estão assim porque é possível utilizar outros parâmetros.

2.3.4 Iniciando uma Rede Complexa

Com o que já foi visto até aqui, já podemos iniciar uma rede com várias máquinas virtuais conectadas entre si. Para isso teremos que usar várias vezes os comandos vistos e executá-los em sequência em um terminal, o que é cansativo. Para facilitar o processo pode-se criar um *script* com os comandos para a criação da rede desejada. Nesse *script* já se pode aproveitar para lançar cada máquina virtual em um terminal diferente usando o Xterm que gera uma janela de linha de comando idêntica há um Terminal Linux, podendo configurar o seu tamanho e nome da janela como visto na Figura 2.10.

```
1 xterm -geometry 60x20 -T nome -e "./<UMLKernel> umid=<nome maquina> mem=128M
      ubda=<COW>,<sistema de arquivos> ethX=daemon,,,/tmp/switch1.ctl" &
```

Figura 2.10: Inicializando uma máquina virtual UML utilizando Xterm.

Assim é possível iniciar uma rede complexa e visualizar cada máquina virtual e executar comandos, cada uma em seu terminal. Conclui-se que utilizar máquinas virtuais UML é uma ótima escolha para realizar experimentos de rede. Entretanto tem-se bastante trabalho e dificuldades para fazer todas as configurações e lançamentos das máquinas virtuais criando um *script* para isso.

Para ser possível utilizar as máquinas UML precisa-se das ferramentas e arquivos já explicados anteriormente. O Xterm já vem instalado por padrão na distribuição Ubuntu usada neste trabalho. Os aplicativos tunctl e UML Switch fazem parte de um pacote de ferramentas para máquinas virtuais UML, chamado UML Utilities, que pode ser instalado no Ubuntu com o seguinte comando ‘‘`apt-get install uml-utilities`’’, sendo usuário *root* em um terminal.

O aplicativo UML Kernel pode ser obtido em `http://uml.devloop.org.uk/index.html`, onde se encontram suas várias versões. E o arquivo que será o sistema de arquivos das máquinas virtuais pode ser obtido em `http://fs.devloop.org.uk/`, onde se encontram vários sistemas de arquivos de diferentes distribuições Linux. No Anexo A é mostrado um exemplo de uma rede utilizando as máquinas virtuais UML.

2.4 Protocolo SNMP

O *Simple Network Management Protocol* (SNMP) é um protocolo lançado em 1988 para atender uma grande necessidade que existia de se fazer o gerenciamento de uma rede complexa,

composta de roteadores, switches e servidores. O SNMP estabeleceu um padrão para o gerenciamento de dispositivos IP e oferece um conjunto de operações para o gerenciamento remoto deles. Esse conjunto de operações pode ser considerado o núcleo do SNMP, permitindo obter e modificar informações em dispositivos que implementam esse protocolo (MAURO; SCHMIDT, 2001).

A estrutura do SNMP pode ser definida a partir de dois tipos de entidades, o gerente e o agente. O gerente atua como o gerenciador da rede, sendo responsável por obter ou alterar as informações dos agentes. Os agentes são os dispositivos gerenciados, os quais rodarão um software que responderão ao gerente quanto houver uma solicitação, fazendo o que for solicitado (MAURO; SCHMIDT, 2001).

Atualmente há definido pela *Internet Engineering Task Force* (IETF), que é a responsável pela definição dos protocolos padrão da Internet, três versões do protocolo SNMP (MAURO; SCHMIDT, 2001):

- SNMP *Version* 1 (SNMPv1) - é a primeira versão do protocolo definido na *Request for Comments* (RFC) 1157.
- SNMP *Version* 2 (SNMPv2) - é a segunda versão do protocolo, onde foram adicionadas novas operações e tipos de mensagens, definido pelas RFCs 1905, 1906 e 1907.
- SNMP *Version* 3 (SNMPv3) - é a terceira versão do protocolo, que tem como foco principal aumentar a segurança na troca de informações entre as entidades gerenciadas, definido pelas RFCs 1905, 1906, 1907, 2571, 2572, 2573, 2574 e 2575.

As informações que o gerente pode obter ou alterar a partir dos agentes são definidas a partir de uma estrutura de informações de gerenciamento (*Structure of Management Information* (SMI)), que define regras para descrever as informações (chamado de objetos) gerenciadas e as permissões que o gerente tem sobre elas. Os objetos gerenciados são definidos utilizando o SMI e depois agrupados em módulos *Management Information Base* (MIB), que pode ser considerado um banco de dados dos objetos gerenciados que o agente rastreia. Qualquer tipo de informação, *status* ou estatísticas que podem ser acessadas pelo gerente em um agente é definido em uma MIB. Resumindo a SMI é o método usado para definir os objetos gerenciados, enquanto a MIB (utilizando a sintaxe da SMI) são os próprios objetos gerenciados que contém as informações (MAURO; SCHMIDT, 2001) (KUROSE; ROSS, 2006).

Os agentes podem implementar várias MIBs diferentes, ou quantas forem necessárias para englobar todas as informações que se queira gerenciar, mas todos os agentes tem que implemen-

tar uma MIB especifica chamada de MIB-II (RFC 1213). Essa MIB padrão define as variáveis para as informações como dados estatísticos de uma interface (velocidade da interface, bytes enviados e recebidos, etc), assim como outras informações relacionadas ao sistemas (local, contato, etc), tendo como objetivo principal fornecer informações gerais sobre o gerenciamento dos protocolos TCP/IP (MAURO; SCHMIDT, 2001).

Os módulos MIB e seus objetos gerenciados são organizados hierarquicamente em árvore, sendo os objetos localizados a partir de um ID (identificador) formado por uma sequência de números inteiros baseados nos nós da árvore, separados por ponto. Utilizando uma estrutura de identificação já padronizada pela *International Organization for Standardization* (ISO), e assim fazendo parte da árvore da linguagem de definições de objetos *Abstract Syntax Notation 1* (ASN.1), como pode ser visto na Figura 2.11 (KUROSE; ROSS, 2006) (MAURO; SCHMIDT, 2001). Como exemplo, o módulo MIB IP, pode ser identificado pelo ID 1.3.6.1.2.1.4, tendo uma grande importância, já que nele estão localizadas as informações sobre o roteamento, entre outras informações.

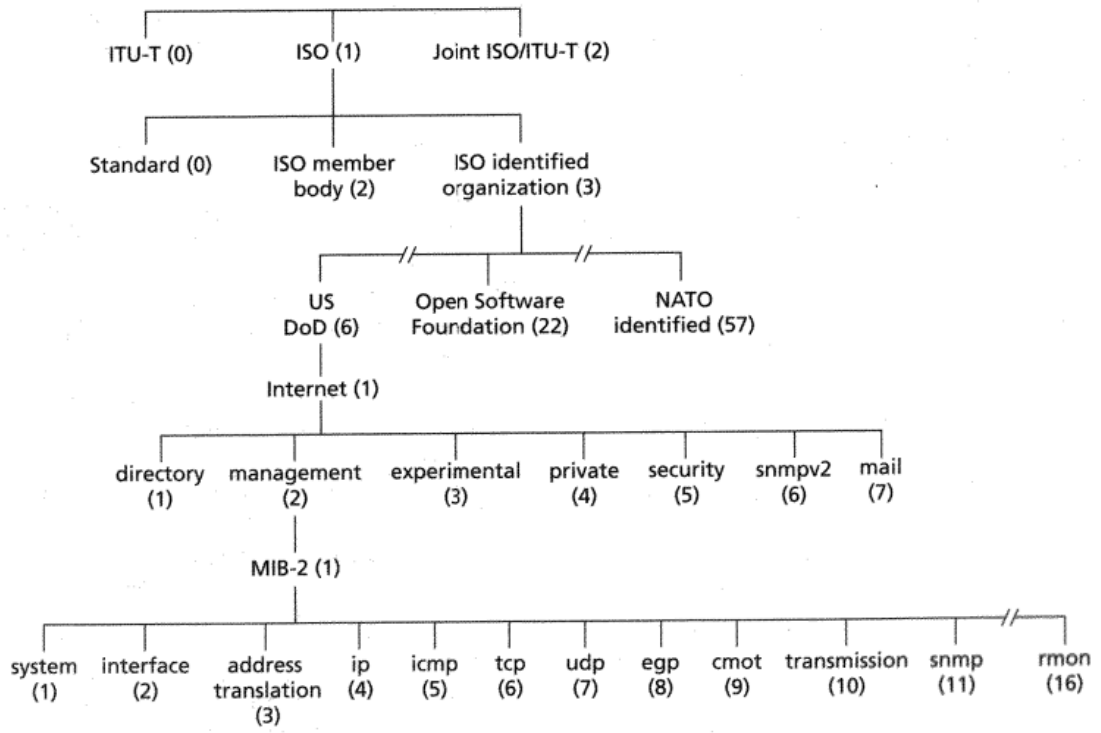


Figura 2.11: Árvore de identificadores de objetos ASN.1 (KUROSE; ROSS, 2006).

As informações (dados) que cada objeto dos módulos MIB guardam podem ser de vários tipos diferentes, definidos pelo SMI. Os principais tipos são (MAURO; SCHMIDT, 2001):

- *Integer*: Número inteiro de 32 bits.

- *Counter*: Um contador de 32 bits que quando atinge seu valor máximo volta ao zero e inicia novamente.
- *Octet String*: Uma string de zero ou mais bytes utilizado para representar strings de texto.
- *IpAdress*: Representa um endereço IPv4 de 32 bits.
- *TimeTicks*: Um número de 32 bits que mede o tempo em centésimos de segundo.
- *Object Identifier*: Uma string decimal composta por pontos que representa o ID de um objeto na árvore de identificação.
- *Sequence*: Define uma lista de tipos diferentes de dados.
- *Sequence of*: Define um objeto que sera formado por um tipo Sequence.

O SNMP utiliza as portas 161 e 162 do protocolo UDP para as trocas de mensagens entre os dispositivos gerenciados. Essas portas são utilizadas como padrão, mas podem ser alteradas. A segurança no SNMPv2 é feito através do conceito de comunidades, que são senhas que serão passadas nas operações realizadas. Existem três tipos de comunidades que são definidas: *read-only*, *read-write* e *trap*. *Read-only* e *read-write* são as senhas das operações do gerente sobre o agente, sendo a primeira a senha para se poder fazer a leitura de uma informação e a segunda para se poder fazer a alteração dessa informação. As próprias definições das MIBs já fazem uso desses conceitos, para definir se elas podem ser apenas lidas ou lidas e alteradas. A *trap* define a senha da única mensagem que o agente manda ao gerente sem nenhuma solicitação sua, também a única que usa a porta UDP 162. A *trap* é usada para o agente comunicar ao gerente que aconteceu algum evento relacionado a ele (uma interface de rede caiu, o agente acabou de ser ligado, etc), e o gerente ao receber essa *trap* irá tomar alguma ação. As mensagens SNMPv2 que realizam essas operações são (MAURO; SCHMIDT, 2001) (KUROSE; ROSS, 2006):

- *GetRequest*: Enviada de gerente ao agente. Faz a leitura de uma ou mais informações de objetos MIB.
- *GetNextRequest*: Enviada de gerente ao agente. Faz a leitura da próxima informação de um objeto MIB de uma lista ou tabela.
- *GetBulkRequest*: Enviada de gerente ao agente. Faz a leitura de um grande bloco de informações.
- *SetRequest*: Enviada de gerente ao agente. Faz a alteração de uma ou mais informações de objetos MIB.

- *InformRequest*: Enviada de gerente ao gerente. Informa a outro gerente informações de um agente que ele não tem acesso.
- *Response*: Enviada de agente a gerente ou gerente a gerente. Mensagem enviada em resposta as requisições das mensagens acima.
- *Trap*: Enviada de agente ao gerente. Informa que ocorreu algum evento relacionado ao agente.

O SNMP define 7 tipos de *traps* diferentes que o agente pode mandar ao gerente. Sendo que seis delas informam um evento já pré-estabelecido e a outra cobre todas os outros eventos que podem ser criados. As *traps* são (MAURO; SCHMIDT, 2001):

- *coldStart*: Indica que o agente foi reiniciado.
- *warnStart*: Indica que o agente reiniciou a si mesmo.
- *linkDown*: Indica que uma interface de rede foi derrubada.
- *linkUp*: Indica que uma interface de rede foi ativada.
- *authenticationFailure*: Indica que alguém tentou consultar o agente com uma comunidade errada.
- *egpNeighborLoss*: Indica que um vizinho do *Exterior Gateway Protocol* (EGP) ficou inativo.
- *enterpriseSpecific*: Indica que é algum outro evento criado.

2.5 Funcionamento do Observador Didático e seus aspectos construtivos

O Observador Didático construído por Farias (2010) foi desenvolvido utilizando as máquinas virtuais descritas na seção 2.3, onde foi construído um cenário composto de computadores e roteadores para se analisar o funcionamento de um protocolo de roteamento nessa rede. A rede desenvolvida pode ser observada na Figura 2.12, juntamente com todas as suas informações.

Como pode ser visto pela Figura 2.12 a rede é composta de seis máquinas virtuais mais o próprio hospedeiro fazendo parte da rede. Entre as máquinas virtuais três delas desempenharão o papel de roteadores, conectados todos entre si através de sub-redes separadas, formando um

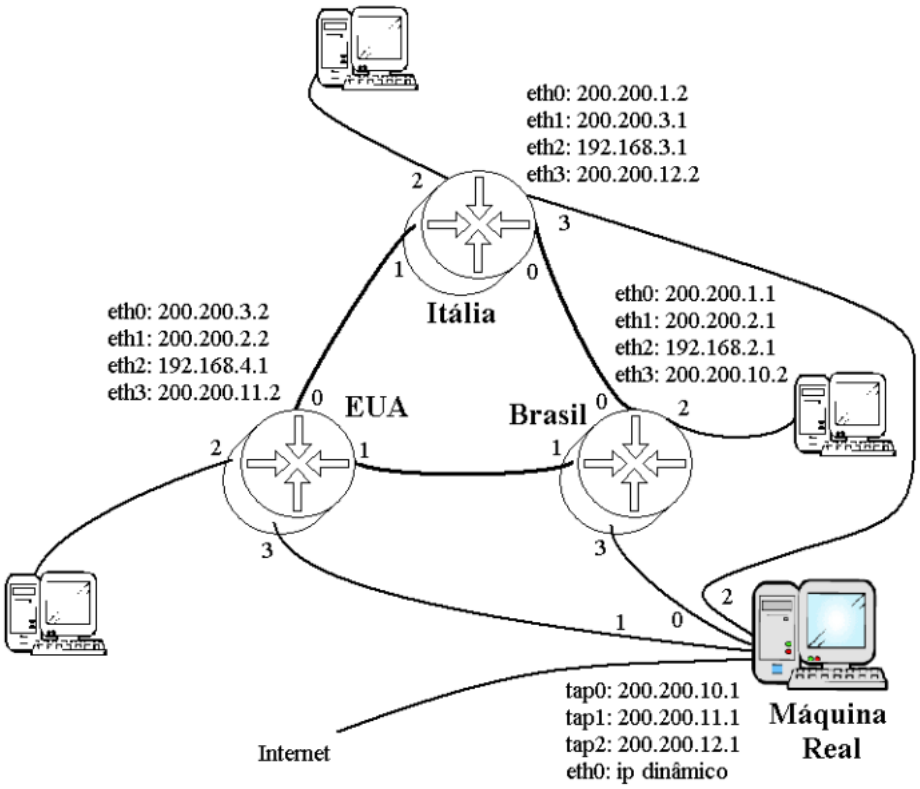


Figura 2.12: Cenário de rede do Observador Didático (FARIAS, 2010).

anel de roteadores no meio da rede, e as outras três máquinas virtuais terão o papel de computadores, cada uma conectada separadamente a cada um dos roteadores. Essa será a rede para se observar o funcionamento dos protocolos de roteamento. O hospedeiro terá três conexões utilizando as interfaces TUN/TAP para se ligar diretamente aos roteadores, para desempenhar o papel de gerente SNMP da rede e prover acesso há internet, mas sem interferir como o protocolo de roteamento que irá funcionar na rede (FARIAS, 2010).

Como o hospedeiro também fará parte da rede sendo o gerente SNMP, terá que ser instalado nele o mesmo *software* já instalado no sistema de arquivos das máquinas virtuais, o Net-SNMP versão 5.4.1, que implementa o protocolo SNMP. Também será necessário que se tenha o IpTables instalado para se fazer as configurações de rede para que as máquinas virtuais que são roteadores acessem a internet.

Para colocar o Observador Didático em funcionamento, fazendo todas as configurações e lançamento das máquinas virtuais, são usados *scripts* que desempenham esses papéis, sendo o Observador Didático composto de *scripts*. Um desses *scripts* é o `redevirtual.sh`, sendo o responsável por fazer toda a construção da estrutura de rede já vista, utilizando os arquivos e ferramentas já descritos na seção 2.3, sendo que o sistema de arquivos das máquinas virtuais é o Ubuntu Server 9.10.

Primeiramente o *script* `redevirtual.sh` copia os arquivos onde serão salvo as tabelas de roteamento do diretório do Observador Didático para o diretório /tmp, copiando também outro *script* chamado `inFile.sh`. Também copia os arquivos snmpd.conf e snmptrapd.conf do diretório do Observador Didático para os respectivos diretórios /etc/snmp/snmpd.conf e /etc/snmp/snmptrapd.conf para fazer a configuração do Net-SNMP no hospedeiro. Em seguida o *script* cria e configura as interfaces TUN/TAP já descritas anteriormente, para os roteadores se conectaram ao hospedeiro. Depois disso é feito todas as configurações no hospedeiro, utilizando também o IpTables, para que ele se torne um roteador, para que os roteadores que são máquinas virtuais acessem a internet (FARIAS, 2010).

Utilizando as ferramentas já descritas na seção 2.3, o *script* `redevirtual.sh` começa a fazer o lançamento das máquinas virtuais. Primeiramente ele inicializa seis *switches* virtuais que são necessários para fazer todas as conexões entre as máquinas. Com a estrutura que irá conectar as máquinas pronta, ele inicia as seis máquinas virtuais que serão usadas no Observador Didático, obtendo assim as as janelas que representam cada máquinas virtual, como pode ser visto na Figura 2.13 (FARIAS, 2010).

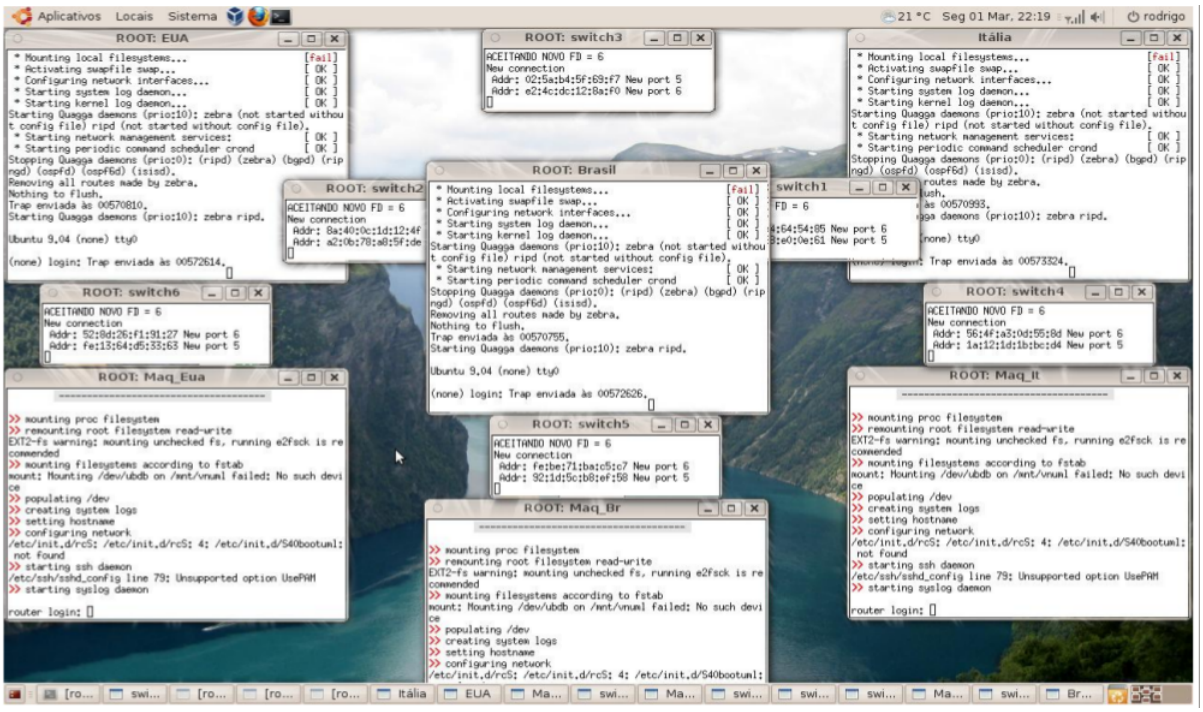


Figura 2.13: Observador Didático em funcionamento (FARIAS, 2010).

O *script* `redevirtual.sh` tem como sua última finalidade encerrar o funcionamento do Observador Didático, se pressionando Ctrl+C no terminal onde o *script* foi executado. Assim ele terminara todos os processos que envolvem as máquinas virtuais, excluindo os arquivos que foram criados e desfazendo as configurações que foram feitas no hospedeiro.

Foi feito uma modificação no arquivo que é o sistema de arquivos das máquinas virtuais por Farias (2010), para serem adicionados dois *scripts* dentro dele. Um desses *scripts* se chama `autoconfi- guracao.sh`, e foi configurado para ser executado na inicialização das máquinas virtuais, ele tem a função de fazer as configurações de redes de todas as máquinas virtuais, e as configurações do Net-SNMP e do protocolo de roteamento *Routing Information Protocol* (RIP), que foi o protocolo escolhido para fazer o roteamento nessa rede, nas máquinas virtuais que são roteadores. Como as máquinas virtuais utilizam o mesmo arquivo para o sistema de arquivos, o *script* tem todas as configurações para todas as máquinas, ele identifica em qual máquinas ele esta rodando e faz as configurações necessárias para aquela máquina (FARIAS, 2010).

O outro *script* que também foi adicionado se chama `observador.sh`, sua função é de ficar monitorando as mudanças na tabelas de roteamento das máquinas virtuais que são roteadores, e quando ele detectar qualquer modificação mandar uma *trap* para o hospedeiro que está atuando como o gerente SNMP da rede para informá-lo. Esse *script* é iniciado apenas nas máquinas virtuais que são roteadores e fica rodando em um *loop* infinito, onde inicialmente ele copia a tabela de roteamento para um arquivo texto, e em sequência fica copiando a atual tabela de roteamento em outro arquivo texto e comparando com a primeira que já foi salva. Quando ele detecta que houve uma mudança, ele atualiza o primeiro arquivo texto com essa tabela e manda a *trap* para o hospedeiro, entrando em *loop* novamente depois disso (FARIAS, 2010).

O hospedeiro ao receber uma *trap* SNMP pode encaminhá-la para um *script*, que irá analisar essa *trap* e tomar as medidas necessárias. Essa é a função do *script* `inFile.sh` que foi copiado junto com os arquivos onde serão salvos as tabelas de roteamento para o diretório /tmp. Esse *script* filtra as *traps* importantes, que são as geradas pelo `observador.sh`, e se for uma *trap* que indica a mudança na tabela de roteamento, ele irá requisitar a tabela de roteamento do roteador que mandou a *trap* e salvar no seu arquivo correspondente (FARIAS, 2010). Então o *script* `inFile.sh` executa por exemplo o comando `‘‘snmpnetstat -On -v2c -c alunos -Crn 200.200.10.2 >> /tmp/Brasil .out’’` e assim captura a tabela de roteamento do roteador Brasil e salva ela no arquivo Brasil.out, juntamente com a hora que ocorreu essa modificação, já que a *trap* quanto é enviado ao hospedeiro contém essa informação. E assim é possível capturar as mudanças de todos os roteadores utilizando o protocolo SNMP, salvando as mudanças de cada roteador em seus respectivos arquivos (FARIAS, 2010).

2.5.1 Resultados obtidos e considerações

O Observador Didático pode ser executado com este cenário de rede desenvolvido e as mudanças nas tabelas de roteamento dos roteadores virtuais ficaram salvas nos arquivos Brasil.out,

Italia.out e EUA.out no diretório /tmp. Se pode acompanhar as modificações nas tabelas de roteamento ao mesmo tempo que o cenário é executado, para isso é preciso visualizar o arquivo onde são salvos as mudanças de uma maneira que ele mostre as modificações que o arquivo sofra. Isso pode ser feito utilizando o comando ‘‘`tail -f /tmp/Brasil.out`’’ por exemplo, para ver as modificações no roteador Brasil, como pode ser visto na Figura 2.14 (FARIAS, 2010).

```
Arquivo  Editar  Ver  Terminal  Abas  Ajuda
root@ULISSES: /home/rodri...   root@ULISSES: ~   root@ULISSES: ~

rodrigo@ULISSES:~$ sudo -i
[sudo] password for rodrigo:
root@ULISSES:~# tail -f /tmp/Brasil.out
200.200.10.0/30            *                   <U>     eth3

Tabela de roteamento modificada às: 0:19:02:27.64
Routing tables
Destination                Gateway             Flags   Interface
default                    200.200.10.1        <UG>    eth3
192.168.2/24               *                   <U>     eth2
200.200.1.0/30             *                   <U>     eth0
200.200.2.0/30             *                   <U>     eth1
200.200.10.0/30            *                   <U>     eth3

Tabela de roteamento modificada às: 0:19:04:11.60
Routing tables
Destination                Gateway             Flags   Interface
default                    200.200.10.1        <UG>    eth3
192.168.2/24               *                   <U>     eth2
192.168.3/24               200.200.1.2         <UG>    eth0
192.168.4/24               200.200.2.2         <UG>    eth1
200.200.1.0/30             *                   <U>     eth0
200.200.2.0/30             *                   <U>     eth1
200.200.3.0/30             200.200.2.2         <UG>    eth1
200.200.10.0/30            *                   <U>     eth3
200.200.11.0/30            200.200.2.2         <UG>    eth1
200.200.12.0/30            200.200.1.2         <UG>    eth0
```

Figura 2.14: Mudanças nas tabelas de roteamento com o Observador Didático em execução (FARIAS, 2010).

Analisando a Figura 2.14 pode-se notar que foram capturadas apenas duas tabelas de roteamento, uma delas logo após a inicialização da máquina, contendo apenas as rotas das redes diretamente ligadas a ela, que é um comportamento normal dos dispositivos IP, e a outro tabela capturada quase 2 minutos depois, com o protocolo de roteamento RIP já estabilizado, adicionando todas as rotas das outras redes não conectadas diretamente ao roteador Brasil. Sabe-se que houve mudanças intermediarias na tabela de rotamento, mas que nem sempre são capturadas devido a velocidade que a tabela de roteamento é atualizada e a velocidade que ela é capturada da máquina virtual.

Com isso pode-se concluir que não é possível acompanhar todas as mudanças nas tabelas de roteamento, devido a captura da tabela de roteamento ser mais lenta do que a velocidade com que ela é atualiza na máquina virtual. Com isso não é possível fazer um estudo mais completo

sobre o funcionamento dos protocolos de roteamento. Sabe-se que algumas tabelas estão sendo perdidas porque o *script* `observador.sh` manda uma mensagem no terminal com a hora que uma *trap* foi enviada, e nota-se o atraso até a tabela aparecer no arquivo .out, se enquanto a tabela não for captura ocorrer uma nova mudança na tabela de roteamento, a mudança anterior será perdida porque será captura a nova mudança e não a anterior que gerou a *trap* (FARIAS, 2010).

O Observador Didático pode ser usado para se fazer estudos de caso. Como, por exemplo, derrubar um enlace entre os roteadores e ver como o protocolo de roteamento se comportará, e quais mudanças acontecerão, como pode ser visto na Figura 2.15, onde o enlace do roteador Brasil com Itália foi derrubado (FARIAS, 2010). Percebe-se que foi capturado uma tabela de roteamento quase 25 minutos depois, onde todas as rotas que usavam a interface eth0 começaram a usar a interface eth1, mas mesmo assim havendo perdas de tabelas, já que há mudanças intermediárias.

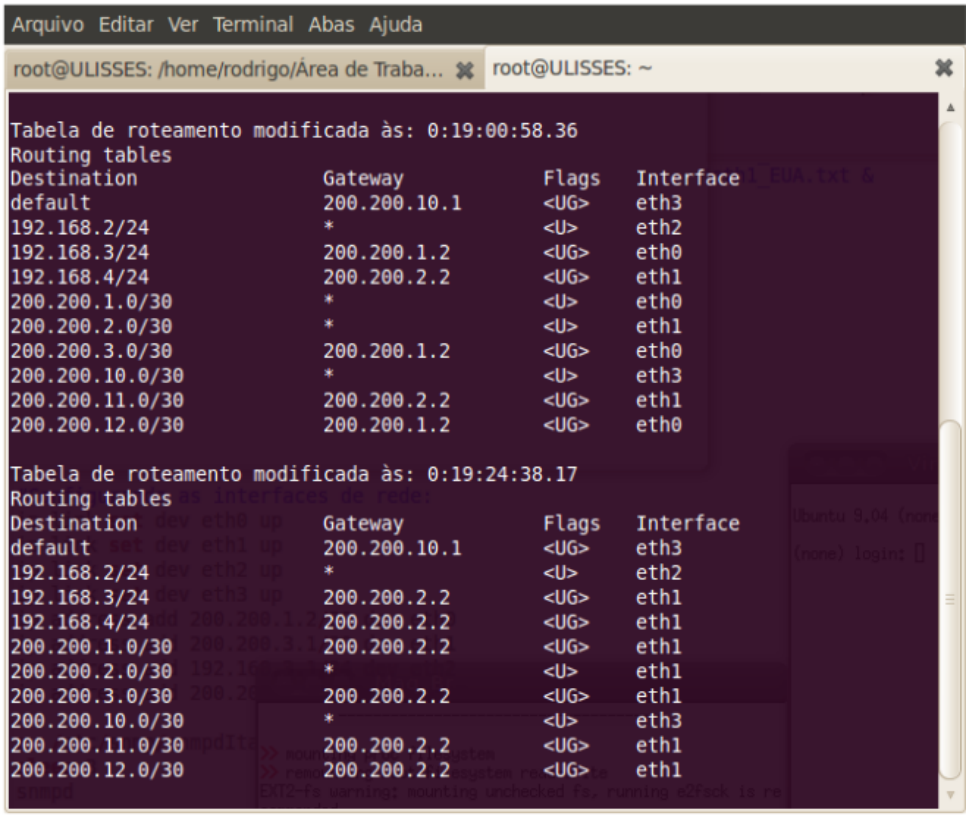


Figura 2.15: Mudança na tabela de roteamento apos o enlace Brasil-Itália ser derrubado (FARIAS, 2010).

Então é possível usar o Observador Didático para se visualizar as tabelas de roteamento dos roteadores da rede descrita, usando-o como um auxilio para o estudo e visualização prática dos protocolos de roteamento, já que nem todas as tabelas são capturadas. Não conseguindo

alcançar seu objetivo inicial, que era de ser usado no ensino dos protocolos de roteamento, porque as tabelas de roteamento são visualizadas em modo texto, sendo necessário abrir os arquivos com as mudanças, e visualizar a sequência das mudanças se observando cada arquivo separadamente, o que não é muito prático. E os *scripts* foram construídos para serem utilizados usando a rede descrita juntamente com o protocolo de roteamento RIP, o que não é muito prático, se for necessário observar outras redes e protocolos, sendo preciso fazer manutenção e modificação desses *scripts*, o que pode ser complicado e difícil se o usuário não tiver familiarização com as máquinas UML, sendo necessário a modificação do sistema de arquivos para a alteração dos *scripts* presentes lá dentro. Com isso percebe-se que pode-se continuar trabalhando no projeto do Observador Didático, fazendo melhorias nele, focando em aumentar sua praticidade e didática para ser usado no ensino dos protocolos de roteamento.

2.5.2 Melhorias pretendidas

As melhorias pretendidas no Observador Didático, visam melhorar a forma de construção dos cenários de rede e também a forma de visualização das modificações nas tabelas de roteamento. Para tal foram escolhidos duas ferramentas o Netkit e o Graphviz. O Netkit tem por finalidade melhorar o aspecto prático para o lançamento e configurações das máquinas virtuais, simplificando o seu uso. Este método substitui os *scripts*, aperfeiçoando a forma de criação dos cenários de rede. Com isso também é possível generalizar a estrutura de rede para qualquer cenário que se queira analisar. Já a ferramenta Graphviz tem como finalidade implementar uma saída gráfica para o Observador Didático, sendo assim possível visualizar as tabelas de roteamento juntamente com a topologia de rede na forma de uma imagem gráfica, tornando mais amigável a exibição das tabelas de roteamento, ao invés de serem visualizadas em formato texto. E com o Observador Didático funcionando juntamente com o Netkit, espera-se fazer testes e usá-lo para observar outros protocolos de roteamento além do RIP. Com o Netkit pode-se fazer a troca de protocolo de roteamento sem a necessidade de grandes modificações em *scripts*.

2.6 Netkit

As redes de computadores são normalmente bastante complexas, compostas por muitos dispositivos, como roteadores, computadores, servidores, etc, cada um contendo várias interfaces de redes. Em cada um desses dispositivos se encontram em execução muitos protocolos. As conexões entre esses dispositivos geram topologias complexas, tornando difícil se fazer exper-

imentos com essas redes, já que os equipamentos de rede são caros ou a rede física disponível não pode ser usada para experimentos, já que ela não pode parar de funcionar (BATTISTA et al., 2007).

Para se fazer experimentos com redes de computadores complexas pode ser usado a simulação ou emulação, que são sistemas que colocam disponível ao usuário um ambiante virtual para experiências, testes e medições. Os sistemas de simulação visam reproduzir o desempenho real do sistema, como perda de pacotes, tempo de transmissão, etc, enquanto sistemas de emulação visam reproduzir precisamente as características de um sistema real, como configurações, protocolos e arquiteturas, com pouca atenção ao desempenho (BATTISTA et al., 2007).

O Netkit pode ser considerado um sistema para emular redes de computadores baseado em máquinas virtuais UML, sendo cada dispositivo de rede emulado uma máquina virtual Linux, baseada no UML Kernel. O sistema operacional Linux dispõem de suporte para a maior parte dos protocolos de rede, podendo ser configurado para funcionar como um roteador ou *switch* (BATTISTA et al., 2007).

O Netkit é composto basicamente por 3 partes, a primeira delas seria um conjunto de ferramentas e comandos que podem ser usados para iniciar uma rede virtual de computadores, a maior parte desses comandos são implementados como *scripts*. A segunda parte seria um sistema de arquivos padrão preparado para criar o sistema de arquivos de cada máquina virtual, sendo que as ferramentas de rede mais usadas já estão instaladas nesse sistema. A terceira parte seria um UML Kernel que será usado como o *kernel* das máquinas virtuais (BATTISTA et al., 2007).

2.6.1 Usando Netkit

Antes de usar o Netkit ele deve ser preparado na máquina hospedeira para ser usado, na seção B.1 do Anexo B é mostrado como fazer essa preparação. O Netkit proporciona dois conjuntos de comandos para serem usados. Os comandos com prefixo V (V comandos) e os comandos com prefixo L (L comandos). Os V comandos atuam como ferramentas de baixo nível para configurar e iniciar máquinas virtuais, enquanto os L comandos proporcionam um ambiente mais fácil para iniciar laboratórios complexos compostos de várias máquinas virtuais, como pode ser visto abaixo (BATTISTA et al., 2007):

- V comandos - Permitem iniciar máquinas com configurações arbitrárias (memória, interfaces de rede, etc)

  - vstart: Inicia uma nova máquina virtual
  - vlist: Exibe uma lista das máquinas virtuais ativas
  - vconfig: Configura as interfaces de rede das máquinas virtuais
  - vhalt: Desliga corretamente uma máquina virtual
  - vcrash Desliga instantaneamente uma máquina virtual
  - vclean: Interrompe todos os processos do Netkit, incluindo as máquinas virtuais e desfaz as configurações feitas no hospedeiro

- L comandos - Permite a fácil implementação de laboratórios complexos composto por várias máquinas virtuais
  - lstart: Inicia um laboratório do Netkit
  - lhalt: Desliga corretamente todas as máquinas virtuais de um laboratório
  - lcrash: Desliga instantaneamente todas as máquinas virtuais de um laboratório
  - lclean: Apaga os arquivos temporários do diretório de um laboratório
  - linfo: Fornece informações sobre um laboratório sem inicializá-lo
  - ltest: Permite executar verificações para checar se o laboratório esta funcionando corretamente

Preparando um laboratório do Netkit

Um laboratório do Netkit é um conjunto de máquinas virtuais pré-configuradas que podem ser inicializadas e desligadas conjuntamente. Pode ser implementado de pelo menos duas maneiras, se criando um *script* que chame os V comandos para iniciar cada máquina virtual, ou criando um laboratório padrão que pode ser iniciado pelos L comandos, que é o mais recomendado.

Para executar um laboratório do Netkit com V comandos, pode ser criado um único *script* que irá chamar o comando `vstart` com algumas opções para iniciar cada máquina virtual. Usando a opção `--exec` do próprio `vstart`, o mesmo *script* pode ser chamado dentro das máquinas virtuais, para ser usado nas próprias configurações de cada máquina, por exemplo, configurar as interfaces de rede. Esse *script* pode verificar se ele está sendo executado no hospedeiro ou em uma máquina virtual, e se estiver dentro de uma máquina virtual, verificar em qual se encontra e fazer as configurações referentes aquela máquina, como pode ser visto no exemplo mostrado na Figura 2.16 (BATTISTA et al., 2007).

```
vstart pc1 --eth0=0 --eth1=1 --exec=this_script
vstart pc2 --eth0=0 --exec=this_script
vstart pc3 --eth0=1 --exec=this_script
if [ `id -u` == "0" ]; then
        case "$HOSTNAME" in
                pc1)
                        ifconfig eth0 10.0.0.1 up
                        ifconfig eth1 10.0.0.2 up;;
                pc2)
                        ifconfig eth0 10.0.0.3 up;;
                pc3)
                        ifconfig eth0 10.0.0.4 up;;
        esac
fi
```

Figura 2.16: *Script* para executar um laboratório do Netkit utilizando os V comandos.

No exemplo mostrado na Figura 2.16, são iniciadas três máquinas virtuais com nomes pc1, pc2 e pc3 conectadas entre si. A opção `--eth0=0` das máquinas pc1 e pc2 fazem com que seja criado uma conexão entra elas nas suas interfaces eth0. O Netkit ao ver essa opção irá automaticamente lançar o UML Switch e fazer as configurações necessárias, o mesmo ocorre com pc1 e pc2 na opção `--eth1=1`, criando uma conexão entre eles nas suas interfaces eth1, sendo lançado assim dois UML Switch. E cada máquina ao executar esse *script* configura automaticamente suas interfaces de rede. Usando o Netkit, neste caso já se torna mais fácil a inicialização das máquinas virtuais e fazer as suas conexões.

Para executar um laboratório do Netkit usando os L comandos, precisa ser preparado uma estrutura que representa esse laboratório. Um laboratório padrão é uma árvore de diretórios que contém vários arquivos e diretórios. Um desses arquivos é o `lab.conf` que descreve a topologia de rede que se quer emular. Os diretórios são um conjunto de sub-diretórios que contém os arquivos de configuração de cada máquina virtual, cada máquina terá um diretório dentro do diretório do laboratório. Outros arquivos são os `.startup` e `.shutdown` que descrevem as ações realizadas por cada máquina quanto são ligadas e desligadas, cada maquina tem os seus arquivos no diretório do laboratório. Pode-se ter também um arquivo opcional `lab.dep`, que contém uma ordem de inicialização das máquinas virtuais, e um diretório opcional `_test` contendo *scripts* para testar se o laboratório está funcionando corretamente (BATTISTA et al., 2007).

O arquivo `lab.conf` além de descrever a estrutura de rede que quer se emular, oferece também algumas outras opções. Para se configurar a estrutura de rede é feito uma lista de configurações utilizando a sintaxe `máquina[arg]=valor`, onde `máquina` é o nome da máquina virtual (por exemplo pc1). Sendo que `arg` pode ser um número inteiro i, onde então `valor` será o nome do domínio de colisão (que será lançado um UML Switch) que a interface ethi deve

ser conectada ou `arg` pode ser uma *string*, onde ela deve ser uma opção do comando `vstart` e `valor` será o argumento da opção (se disponível). Um exemplo de um arquivo `lab.conf` juntamente com a topologia de rede gerada pode ser visto na Figura 2.17, onde foi descrida uma rede composta de três roteadores conectados entre si através de dois domínios de colisões. No pc2 foi utilizado uma opção do `vstart` para se configurar a máquina com 256 MB de memória virtual.

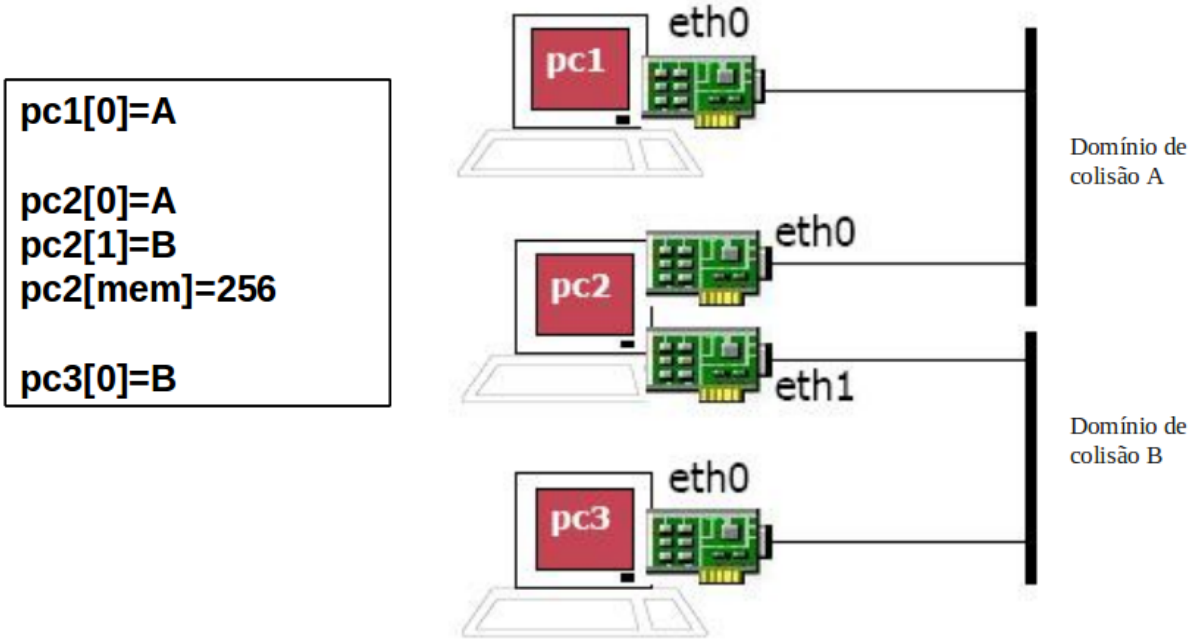


Figura 2.17: Exemplo de arquivo `lab.conf` e a topologia gerada (BATTISTA et al., 2007).

Mais uma configuração que pode ser usado no `lab.conf` e é opcional tem a sintaxe `machines=‘‘pc1 pc2 pc3...’’`, que declara formalmente as máquinas virtuais que fazem parte do laboratório, já que por padrão a existência de um sub-diretório dentro do diretório do laboratório significa que será inicializada uma máquina virtual com o mesmo nome do sub-diretório, sendo que o Netkit inicializa uma máquina virtual para cada sub-diretório, a menos que a opção `machines=‘‘pc1 pc2 pc3...’’` seja usada.

Quanto uma máquina virtual é iniciada utilizando o Netkit, todo o conteúdo do sub-diretório com o nome da máquina virtual no diretório do laboratório é mapeado (copiado) para a raiz (/) do sistema de arquivos da máquina virtual. Por exemplo, o arquivo em pc1/etc/resolf.conf será copiado para /etc/resolf.conf na máquina virtual pc1. Isso só acontecerá a primeira vez que a máquina for inicializada, para se fazer de novo a cópia é necessário apagar o sistema de arquivos da máquina virtual, os arquivos `.disk`, que são os arquivos criados automaticamente pelo Netkit para utilizar o mecanismo COW explicado na seção 2.3.2.

Com esse recurso de copiar os arquivos para a máquina virtual, se torna muito mais fácil a

configuração e utilização das mesmas, facilitando a configuração de serviços e a transferência de arquivos importantes para a máquina, sem se precisar editar o arquivo do sistema de arquivos. Outra facilidade que o Netkit implementa, depois que um laboratório é inicializado, é a criação de dois diretórios na máquina virtual que apontam para os diretórios *home* do usuário e do laboratório no sistema de arquivos do hospedeiro, sendo permitida a leitura e escrita nos mesmos. Assim, é desta forma que a máquina virtual pode acessar e utilizar os arquivos presentes no hospedeiro e criar arquivos nele. Isto é importante se precisa ser salvo alguma informação, podendo ser salva diretamente no hospedeiro. Os diretórios criados para isso na máquina virtual são `/hosthome` e o `/hostlab`.

Os arquivos `.startup` e `.shutdown` não são obrigatórios, mas se existirem eles terão o nome da máquina virtual referentes a eles na frente do ponto. Esses arquivos podem ser considerados *scripts* que irão dizer o que as máquinas virtuais devem fazer quanto forem iniciadas e desligadas, sendo executados dentro das máquinas virtuais. Outros dois arquivos opcionais que podem existir são o `shared.startup` e `shared.shutdown`, que tem o mesmo proposito dos outros arquivos, só que afetam todas as máquinas virtuais. Na inicialização das máquinas virtuais é executado primeiro o arquivo referentes a todas as máquinas para depois executar o seu próprio, e no desligamento das máquinas ocorre o contrário, sendo executado primeiro o referente a sua própria máquina para depois executar o referente a todas as máquinas. Um uso importante para os arquivos `.startup` é configurar as interfaces de rede e iniciar os serviços que serão usadas na máquina (BATTISTA et al., 2007).

Pode-se iniciar várias máquinas virtuais ao mesmo tempo usando a opção -p do comando `lstart`, que é a inicialização em paralelo. Mas pode-se querer que certa máquina virtual só inicialize depois que outra já estiver inicializada, para isso pode-se usar um arquivo chamado `lab.dep` dentro do diretório do laboratório que descreve a ordem de inicialização das máquinas. Por exemplo, pc3 só pode inicializar depois que pc2 e pc1 já estiverem inicializados, para isso se coloca o comando `pc3: pc2 pc1` dentro do arquivo `lab.dep` (BATTISTA et al., 2007).

Com a estrutura do laboratório do Netkit descrita acima já pronta, pode se iniciar as máquina virtuais utilizando os L comandos. Para isso o mais recomendado é entrar no diretório do laboratório e executar o comando que se queira, por exemplo, `lstart`, e assim serão iniciadas todas as máquinas do laboratório. Opcionalmente pode se passar uma lista das máquinas virtuais como parâmetro que serão afetados pelo comando.

O Netkit pode ser considerado uma ferramenta para a utilização das máquinas virtuais UML e para realização de experimentos, facilitando a utilização para o usuário com os seus mecanismos já implementados. Deve-se ressaltar que não foram vistos todas as opções disponíveis

do Netkit, como por exemplo, fazer conexões utilizando as interfaces TUN/TAP, mas podendo ser realizado com apenas um único comando e automaticamente criando a interface e depois removendo ela no hospedeiro. Para quem não tem o conhecimento sobre as máquinas UML e os `scripts` que precisam ser criados para a sua inicialização, o Netkit foi criado para facilitar isso. Na seção B.2 do Anexo B é mostrado um exemplo de uma rede utilizando máquinas virtuais através da ferramenta Netkit.

2.7 Graphviz

Graphviz é um pacote de ferramentas de código aberto iniciado pela AT&T Labs Research para desenhar grafos, sendo a abreviatura de Graph Visualization Software. Fornece também bibliotecas para que outras aplicações e softwares utilizem suas ferramentas, sendo um software livre (GRAPHVIZ. . . ,).

Sendo o Graphviz uma ferramenta para desenhar grafos, veremos alguns conceitos sobre eles. Um grafo é representado tipicamente como um conjunto de pontos (vértices) ligados por retas (as arestas), que dependendo da aplicação as arestas podem ser direcionais e são representadas por setas. Um exemplo de grafo é mostrado na Figura 2.18. Vértices e/ou arestas podem ter um peso/custo (numérico) associados a eles.

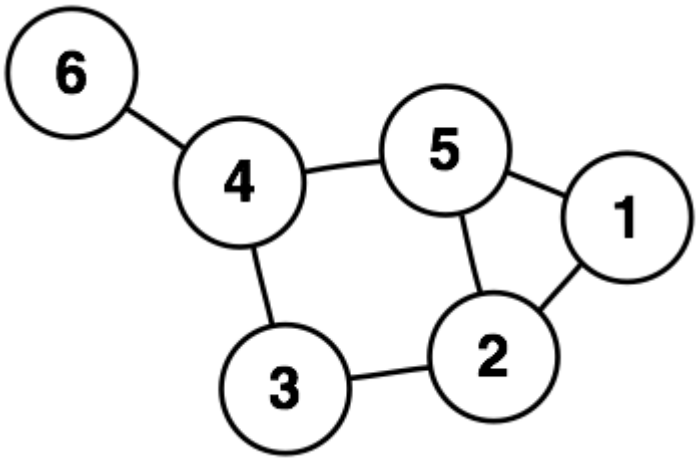


Figura 2.18: Um grafo com 6 vértices e 7 arestas.

Grafos que não são direcionados, tendo apenas linhas retas ligando seus vértices, são chamados simplesmente de grafos. Os grafos direcionados, que são representados por arestas com setas indicando o sentido das ligações dos vértices, são chamados de dígrafos. Os grafos são representados tipicamente se desenhando um círculo para os vértices, e para as aresta é desenhado um arco conectando os vértices. A representação gráfica do grafo, não deve ser con-

fundida com o grafo em si, já que várias diferentes imagens podem corresponder ao mesmo grafo, o que importa é quais vértices estão conectados entre si por quantas arestas.

Os grafos são de grande importância, porque estruturas que podem ser representadas por grafos estão em toda parte. Como muitos problemas de interesse prático como questões sobre certos grafos que podem ser representados. Por exemplo, pode-se representar um mapa de estradas através de grafos e usar algoritmos específicos para determinar o caminho mais curto entre dois pontos; usados para representar máquinas de estados finitos e fluxogramas; também nas redes de computadores, sendo cada terminal representado por um vértice, o cabo de redes pelas arestas e o custo associado a latência.

O Graphviz tem importantes aplicações em redes, bioinformática, engenharia de software, banco de dados e web design. Existem inúmeros programas e ferramentas que utilizam o Graphviz em diferentes áreas. As imagens são geradas a partir de um arquivo de texto simples que contém a descrição do grafo, podendo ser gerado imagens em vários formados diferentes. Graphviz tem muitos recursos e opções úteis para uma grande personalização e criação de grafos, tais como cores, fontes, leiautes dos nós (vértices), estilo de linhas e outros.

Graphviz é composto por cinco programas que geram imagens de diferentes leiautes para um mesmo arquivo texto com a descrição do grafo. Esses programas são: `dot`, `neato`, `fdp`, `twopi` e `circo`, sendo que cada um utiliza algoritmos diferentes para a criação das imagens, gerando imagens diferentes entre si para um mesmo grafo. No Anexo C é mostrado como usar a ferramenta Graphviz e tem-se um exemplo de um grafo mais complexo, mostrando o arquivo texto que descreve o grafo e a sua imagem gerada.

3 *Implementação das melhorias*

Neste capítulo será descrito como foi realizado as implementações das melhorias no Observador Didático, depois de realizado o estudo apresentado no capítulo 2, para ter-se os conhecimentos necessários para executar as implementações.

3.1 Implementação do Observador Didático utilizando o Netkit

Para implementar o Observador Didático utilizando o Netkit inicialmente foi feito um laboratório com o Netkit utilizando o mesmo cenário de rede da Figura 2.12, sendo feito as adaptações e mudanças necessárias. Uma das mudanças que difere bastante do Observador Didático já construído é que o hospedeiro que atuava como o gerente SNMP da rede, sendo necessário ligações com as máquinas virtuais, foi substituído por uma máquina virtual. Isso é possível já que a máquina virtual que será o gerente SNMP, pode salvar os resultados das mudanças de roteamento no próprio hospedeiro, pelas facilidades que o Netkit implementa. Outro motivo para que o gerente SNMP da rede fosse o hospedeiro, era para que ele fornecesse internet para as máquinas virtuais que são roteadores, entretanto concluiu-se que não era necessário uma vez que as máquinas virtuais não utilizam a internet. Todavia é possível, se necessário, ligar a máquina virtual que será o gerente SNMP ao hospedeiro para prover a internet as máquinas virtuais, nesse caso haverá somente uma ligação com o hospedeiro.

Desta forma foi montada a estrutura que será o laboratório do Netkit, que implementa o Observador Didático. Para isso foi criado um diretório e dentro dele foram criados outros subdiretórios, ao todo sete, um para cada máquina virtual necessária, ao todo são três roteadores, três computadores e mais o gerente SNMP. Para definir as conexões entre as máquinas virtuais foi criado o arquivo `lab.conf`, sendo de fácil configuração a topologia de rede necessária. No arquivo `lab.conf` está a configuração da máquina virtual que será o gerente SNMP conectada as máquinas virtuais que serão os roteadores. Também foram configuradas no arquivo

`lab.conf` as máquinas virtuais que fazem parte do laboratório. O arquivo criado pode ser visto na Figura 3.1.

```
machines="router_brasil router_eua router_italia host_brasil host_eua
    host_italia gerente_snmp"

host_brasil[0]="brasil"

host_eua[0]="eua"

host_italia[0]="italia"

router_brasil[0]="br-it"
router_brasil[1]="br-eua"
router_brasil[2]="brasil"
router_brasil[3]="gerente"

router_eua[0]="eua-it"
router_eua[1]="br-eua"
router_eua[2]="eua"
router_eua[3]="gerente"

router_italia[0]="br-it"
router_italia[1]="eua-it"
router_italia[2]="italia"
router_italia[3]="gerente"

gerente_snmp[0]="gerente"
```

Figura 3.1: Arquivo `lab.conf` criado.

Nesse atual estado já poderíamos iniciar o laboratório e teríamos as sete máquinas virtuais funcionando conectadas entre si através dos UML Switch que o Netkit lança automaticamente, mas não haverá comunicação entre as máquinas, porque suas interfaces de rede não foram configuradas. O *script* `autoconfiguração.sh`, que desempenha o papel de configurar as máquinas virtuais no Observador Didático construído por Farias (2010), no Netkit pode ser substituído pelos arquivos `.startup`, se tornando mais prático a configuração de cada máquina virtual. Foi criado um arquivo `.startup` para cada máquina virtual, onde é configurado as interfaces de rede e inicializado os serviços necessários. Nos roteadores é lançado os serviços referentes ao protocolo de roteamento e ao SNMP, no gerente é lançados os serviços referentes ao SNMP e o serviço do SNMP responsável por receber as *traps* e tratá-las e nos computadores não é inicializado serviço nenhum.

Outro papel do `script autoconfiguração.sh` é configurar os serviços lançados, criando os arquivos necessários para sua configuração. No Netkit não é preciso alterar o sistema de arquivos ou criar *scripts* para criar os arquivos de configuração. Os arquivos de configuração só precisam ser criados e colocados nos sub-diretórios referentes a cada máquina virtual, já que

quanto são inicializadas, esses arquivos são copiados para o sistema de arquivos das máquinas virtuais, somente precisa ser criado a mesma hierarquia de diretórios onde os arquivos precisam ser colocados.

Nas máquinas virtuais que são roteadores, foi adicionando o `script monitor.sh` nos seus sub diretórios, que é o *script* `observador.sh` modificado, responsável por identificar as mudanças nas tabelas de roteamento. Nos arquivos `.startup` dos roteadores foi adicionado esse *script* para ser iniciado. A modificação realizada no *script* `observador.sh` é referente ao método de identificação que detecta uma mudança na tabela de roteamento.

O *script* `inFile.sh` responsável por analisar as *traps* SNMP recebidas foi colocado dentro do sub-diretório da máquina virtual do gerente SNMP, juntamente com os arquivos de configuração do Net-SNMP. Esse *script* foi modificado para salvar as tabelas de roteamento capturadas no próprio diretório do laboratório do Netkit, em um sub-diretório chamado `resultados`, que precisa ser criado dentro do diretório do laboratório. Por essa razão, tem que ser explicitamente configuradas quais são as máquinas virtuais do laboratório no arquivo `lab.conf`, senão será iniciada uma máquina virtual chamada resultados.

Como o gerente SNMP também será uma máquina virtual, sendo o responsável por capturar as tabelas de roteamento das máquinas virtuais que são roteadores, é necessário que ele já esteja funcionando antes das outras máquinas virtuais que compõem a rede sejam inicializadas. Para isso pode ser usado o arquivo `lab.dep`, que será criado dentro do diretório do laboratório, descrevendo a ordem de inicialização das máquinas virtuais. Assim o gerente SNMP será inicializado primeiro e só depois as outras máquinas serão inicializadas, o arquivo `lab.dep` pode ser visto na Figura 3.2.

```
1 router_brasil router_eua router_italia host_brasil host_eua host_italia:
      gerente_snmp
```

Figura 3.2: Arquivo `lab.dep` criado.

Com a estrutura do laboratório do Netkit pronta, o Observador Didático está pronto para ser usado juntamente com o Netkit. A estrutura do laboratório pode ser visto na Figura 3.3. Para iniciar o laboratório, é preciso executar o comando `lstart` estando no diretório do laboratório. Desse modo será inicializadas as máquinas virtuais e o Observador Didático entrará em execução capturando as tabelas de roteamento, salvando-as no diretório `resultados`. As máquinas virtuais do laboratório em funcionamento podem ser vistas na Figura 3.4, destacando-se as máquinas virtuais que são o gerente SNMP e os roteadores, podendo ser visto nos roteadores as mensagens indicando que eles mandaram *traps* SNMP para o gerente.

Figura 3.3: Estrutura do Observador Didático utilizando o Netkit.

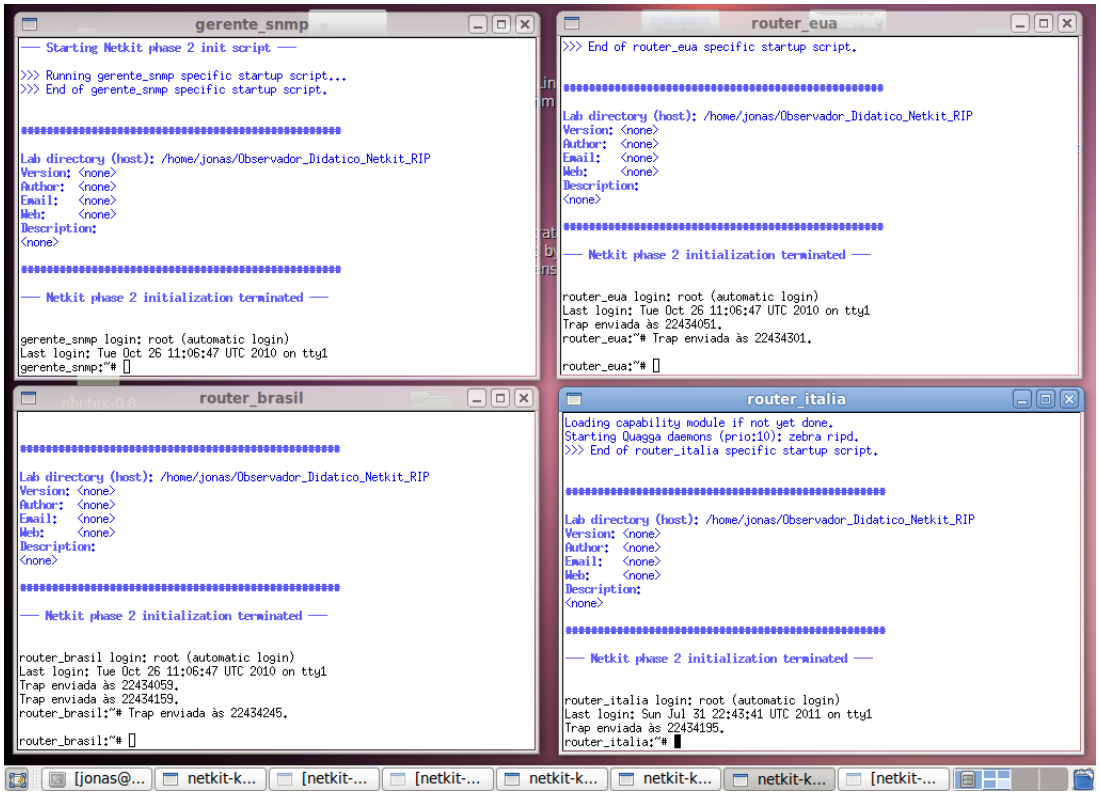


Figura 3.4: Observador Didático utilizando o Netkit em funcionamento.

No Anexo D se encontra um documento que descreve como preparar um laboratório do Netkit para ser usado junto com o Observador Didático, cumprindo um dos objetivos que é generalizar a estrutura física utilizada para poder observar outros cenários de rede. Esse Anexo também auxilia no entendimento da implementação do Observador Didático utilizando o Netkit.

3.2 Problema encontrado com o Net-SNMP

Com o Observador Didático implementado com o Netkit, realizou-se testes para ver se seu funcionamento era correto, capturando as tabelas de roteamento. Em um primeiro momento, percebeu-se que o Observador Didático funcionava corretamente, capturando as tabelas de roteamento e as salvando.Também verificou-se que o Observador Didático ficou mais rápido, capturando as tabelas mais rapidamente. Entretanto depois de uma análise mais detalhada das tabelas salvas, notou-se que, durante um mesmo período de tempo, a mesma tabela era sempre salva, mesmo recebendo a *trap* que a tabela mudou, a tabela que retornava era a mesma. Checando na máquina virtual que é o roteador a tabela realmente tinha mudado, e executando manualmente o comando para pegar a tabela pelo protocolo SNMP, a tabela que retornava era mesmo a tabela antiga. Depois de mais alguns testes e discussões concluiu-se que que o software utilizado para implementar o protocolo SNMP no gerente e nos agentes, que é o Net-SNMP, um projeto de código livre, estava fazendo cache das tabelas de roteamento por aproximadamente dois minutos, e nesse período é retornado a mesma tabela. Várias outras informações também fazem cache por certos períodos de tempo, como as informações sobre as interfaces de rede, que retorna o mesmo número de pacotes recebidos pela interface. Esse mecanismo de se fazer cache das informações nos agentes é comum nos *softwares* que implementam o protocolo SNMP, com o objetivo de não sobrecarregar e roubar recursos dos dispositivos gerenciados, que são equipamentos importantes, que não podem ser prejudicados tendo que atualizar constantemente suas informações (MAURO; SCHMIDT, 2001).

Isso foi um grande problema para o o Observador Didático, já que capturar a mesma tabela de roteamento por dois minutos, compromete o seu objetivo que é acompanhar a evolução das tabelas de roteamento, uma vez que nesse tempo podem ocorrer várias mudanças nas tabelas. Primeiramente, para resolver esse problema, foi checado se podia ser um problema da versão do Net-SNMP usada, uma vez que o sistema de arquivos do Netkit, que usa a distribuição Debian do Linux, já vem com o Net-SNMP instalado. Mas, observou-se que a versão do Net-SNMP utilizada no Observador Didático construído por Farias (2010) era a mesma usada no Observador Didático construído com o Netkit. Foram então realizados testes no Observador Didático construído por Farias (2010) e constatou-se o mesmo problema.

Após confirmar que não era um problema da versão utilizada, e que fazer cache das informações é seu funcionamento normal, tentou-se encontrar alguma configuração nos arquivos do Net-SNMP que desativasse o cache, mas não foi encontrada nenhuma configuração para isso. Conti-nuou-se tentando encontrar alguma solução, e foi encontrado nos agentes uma MIB que contêm configurações do Net-SNMP, podendo ser lido e alterados essas configurações através das mensagens do SNMP que fazem isso. Essa MIB pode ser vista na Figura 3.5, notando-se inúmeras informações e configurações referentes ao cache.

```
NET-SNMP-AGENT-MIB::nsModuleTimeout."".12.1.3.6.1.6.3.16.1.5.2.1.6.127 = INTEGER
: -1
NET-SNMP-EXTEND-MIB::nsExtendNumEntries.0 = INTEGER: 0
NET-SNMP-AGENT-MIB::nsCacheDefaultTimeout.0 = INTEGER: 5
NET-SNMP-AGENT-MIB::nsCacheEnabled.0 = INTEGER: false(2)
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.2.2 = INTEGER: 15
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.4 = INTEGER: 5
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.4.24.4 = INTEGER: 60
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.4.24.7 = INTEGER: 60
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.4.31.1 = INTEGER: 60
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.4.34 = INTEGER: 30
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.4.35 = INTEGER: 60
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.5 = INTEGER: 5
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.6 = INTEGER: 5
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.6.13 = INTEGER: 5
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.6.19 = INTEGER: 60
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.6.20 = INTEGER: 60
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.7 = INTEGER: 5
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.7.5 = INTEGER: 5
NET-SNMP-AGENT-MIB::nsCacheTimeout.1.3.6.1.2.1.7.7 = INTEGER: 60
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.2.2 = INTEGER: cached(4)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.4 = INTEGER: expired(5)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.4.24.4 = INTEGER: cached(4)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.4.24.7 = INTEGER: cached(4)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.4.31.1 = INTEGER: cached(4)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.4.34 = INTEGER: cached(4)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.4.35 = INTEGER: cached(4)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.5 = INTEGER: expired(5)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.6 = INTEGER: expired(5)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.6.13 = INTEGER: expired(5)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.6.19 = INTEGER: cached(4)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.6.20 = INTEGER: cached(4)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.7 = INTEGER: expired(5)
NET-SNMP-AGENT-MIB::nsCacheStatus.1.3.6.1.2.1.7.5 = INTEGER: expired(5)
```

Figura 3.5: MIB do Net-SNMP contendo configurações dos agentes.

Tentou-se através das mensagens *SetRequest* do SNMP modificar manualmente todos os valores contidos nessa MIB para tentar desativar o cache. Um a um esses valores foram modificados e depois realizados testes. Com essas modificações algumas informações deixaram de realizar cache, como a informação do número de pacotes recebidos pela interface de rede, que agora a cada requisição vinha com um número maior, mas as tabelas de roteamento continuaram fazendo cache. Continuou-se tentando modificar outros valores definidos em outras MIBs e encontrar algum comando que pudesse ser executado nos agentes que desativasse o cache, mas não se obteve resultados positivos.

Como ultima solução e mais complicada, foi pensado em alterar o código fonte do Net-SNMP para desativar o cache, já que ele é de código livre. No site do projeto Net-SNMP `http://www.net-snmp.org/`, se tem disponível para fazer o download do código fonte de suas várias versões, que são programadas na linguagem de programação C, e há também um grupo de discussão sobre o seu desenvolvimento. Começou-se então a fazer alterações no código fonte para desativar o cache, depois de muito esforço e várias tentativas foi obtido sucesso na desativação do cache. Ressalta-se que esta tarefa demandou muito trabalho e dificuldades, pois envolveu mexer em um grande projeto como o Net-SNMP, o qual chega há ter quase dois mil arquivos de códigos fontes espalhados em vários sub-diretórios.

A alteração do código fonte Net-SNMP foi realizada no sistema de arquivos do Netkit, sendo preciso primeiro remover a versão do Net-SNMP já instalada e depois passar o diretório do projeto do Net-SNMP para o sistema de arquivos do Netkit para ser compilado com as modificações. Para poder fazer a compilação do Net-SNMP foi preciso ainda que outras ferramentas fossem instaladas e que fossem salvas as modificações no sistema de arquivos do Netkit, antes que fosse inicializado o laboratório do Observador Didático. Com apenas um comando, que é `‘‘vstart -W --mem=128 --eth0=tap,200.200.10.1,200.200.10.2 host’’` (precisa ser usuário *root*), pode ser inicializada uma máquina virtual com acesso a internet para poder-se instalar programas e salvar as modificações feitas no sistema de arquivos do Netkit.

Depois de várias tentativas conseguiu-se desativar o cache que era feito das tabelas de roteamento. Em seguida foram realizados testes e comprovou-se que as tabelas capturadas pelo protocolos SNMP eram as mesmas que estavam atualmente nas máquinas virtuais que são roteadores. Para o funcionamento correto do Observador Didático precisa ser usado o sistema de arquivos do Netkit modificado, que contêm o Net-SNMP alterado.

3.3 Implementação de uma saída gráfica para o Observador Didático

Com o Observador Didático funcionando juntamente com Netkit é possível observar as mudanças nas tabelas de roteamento. Contudo as mudanças são visualizadas em modo texto, sendo necessário a abertura dos arquivos no diretório `resultados` dentro do diretório do laboratório do Netkit. Utilizando-se da ferramenta Graphviz é possível gerar uma saída gráfica para o Observador Didático, representando a topologia de rede na forma de um grafo, juntamente com as tabelas de roteamento de cada roteador.

Com a grande quantidade de recursos e opções para a criação e personalização dos grafos,

foi realizado vários testes e tentativas com o Graphviz para obter uma imagem que representasse a topologia de rede juntamente com as tabelas de roteamento, realizando um trabalho bastante prático. Conseguiu-se imagens que tiveram um bom resultado para a representação da topologia de rede juntamente com as tabelas, inclusive colocando figuras de roteados e computadores nos vértices do grafo, indicação das interfaces de rede nos roteadores e seus nomes, como pode ser visto na Figura 3.6.

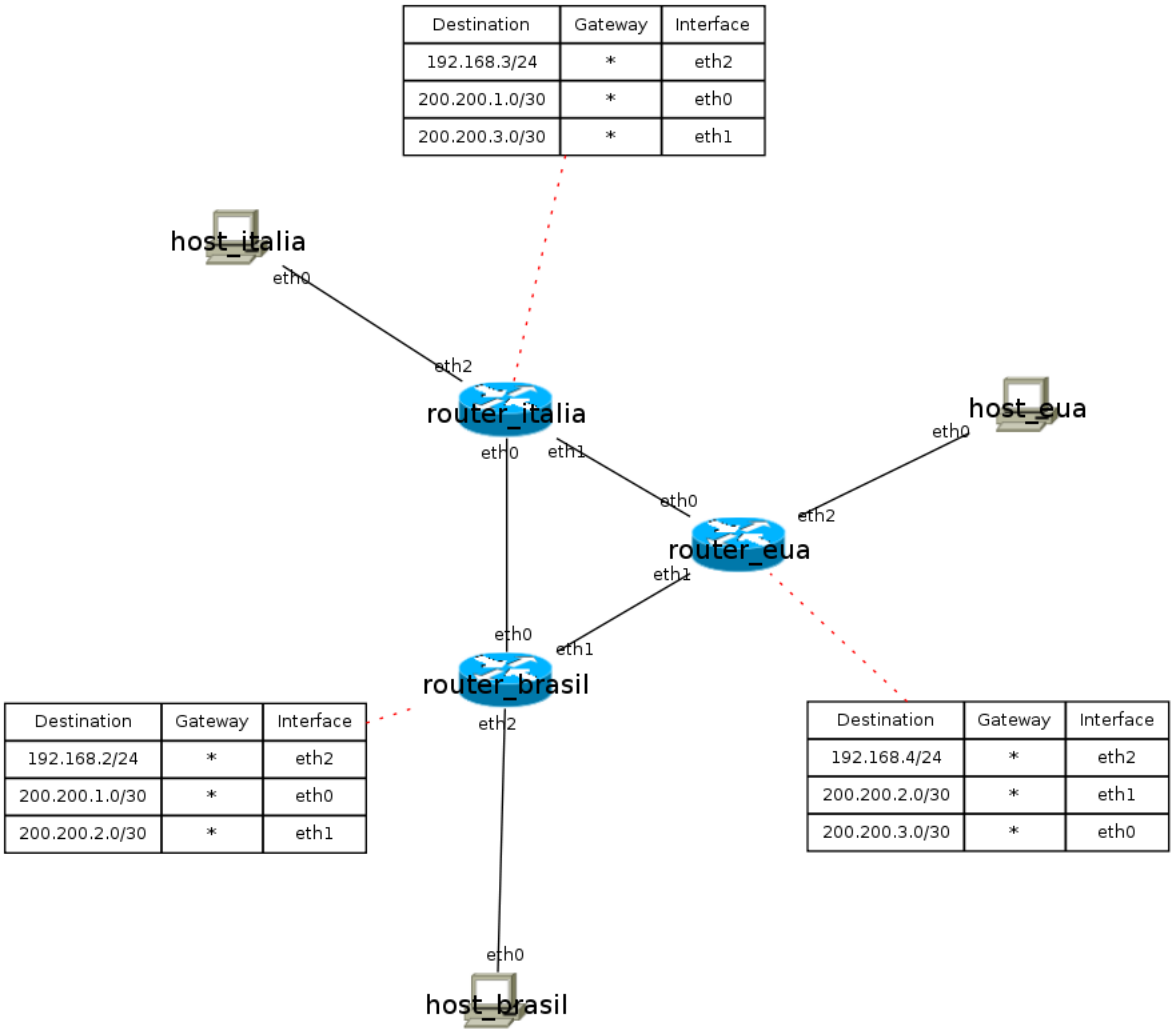


Figura 3.6: Imagem da topologia de rede e tabelas de roteamento gerada com a ferramenta Graphviz.

Optou-se por utilizar as opções e configurações mais genéricas no arquivo texto que descreve o grafo, podendo assim gerar a imagem a partir de qualquer um dos cinco programas que compõem o Graphviz, evitando usar opções que só trarão resultados em se utilizar certo programa do Graphviz. Foi utilizadas opções que permitiram personalizar os vértices do grafo, que representam os nós da rede, colocando figuras de roteadores e computadores. Utilizou-se opções para colocar os nomes das interfaces de rede em cada uma das ponta das arestas,

que representam as ligações entre os nós da rede. As tabelas de roteamento são vértices do grafo, que foram usadas opções para apareceram como tabelas, e se utilizou arestas pontilhadas na cor vermelha para fazer a ligação dos vértices que representam os roteadores com os vértices que representam as tabelas de roteamento. No caso de haver mais de dois computadores/roteadores em um mesmo domínio de colisão, é possível fazer essas ligações conectando esses computadores/roteadores há um vértice que não representará nada para a topologia de rede, sendo esse vértice configurado para que apareça de um tamanho muito pequeno, quase imperceptível, dando a impressão que os computadores/roteadores estão ligados diretos uns aos outros pelas arestas, como pode ser visto na Figura 3.7, para ligar os computadores Itália e Itália 2 ao roteador Itália.

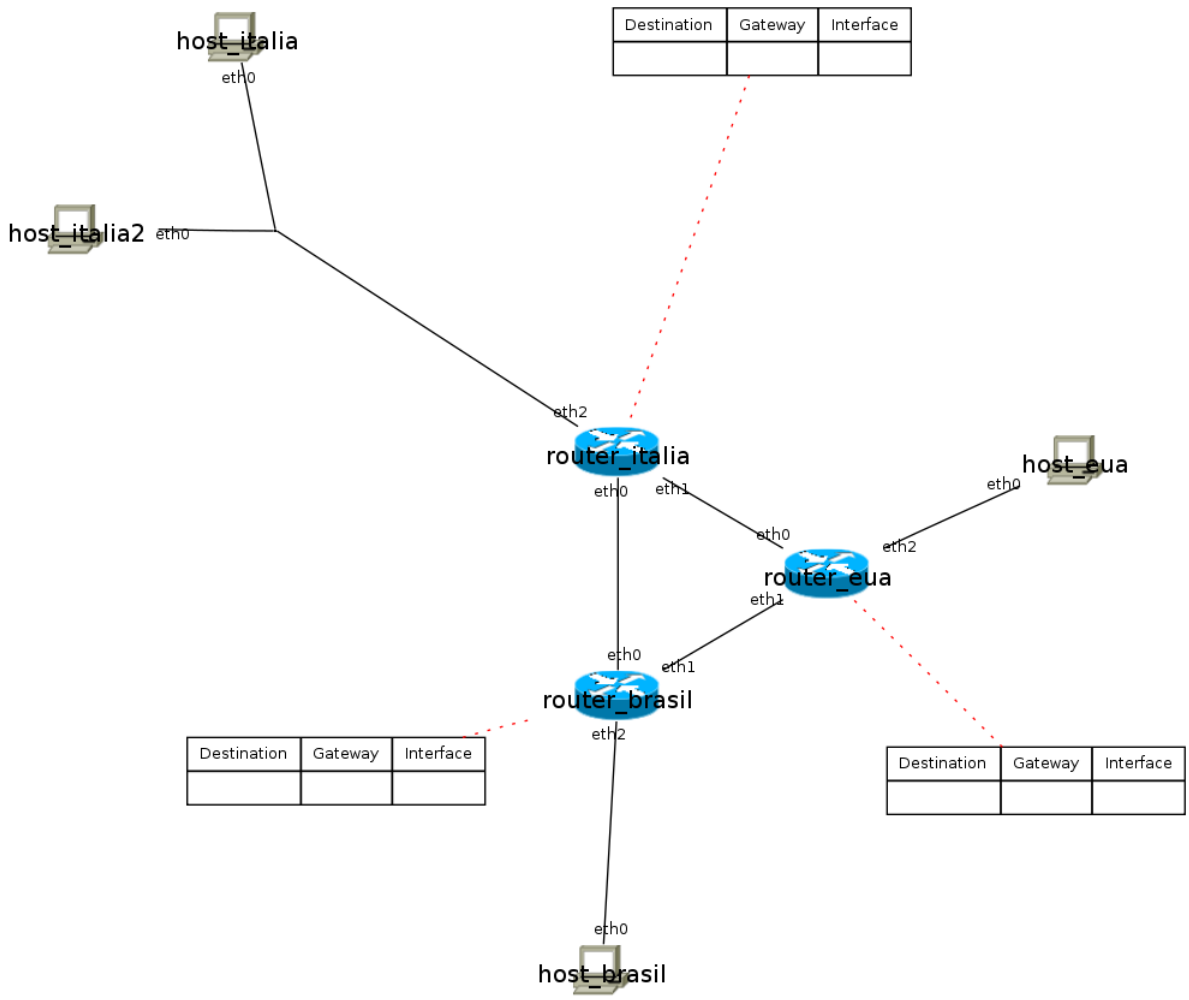


Figura 3.7: Imagem gerada com a ferramenta Graphviz representando três nós de rede em um mesmo domínio de colisão.

É possível utilizando a ferramenta Graphviz visualizar as tabelas de roteamento de forma gráfica. Mas as Figuras 3.6 e 3.7 foram geradas manualmente, olhando as tabelas de roteamento salvas e gerando o código apropriado para o Graphviz gerar aquele figura. Foi preciso

implementar um programa que faça isso para o usuário, que automaticamente olhe esses arquivos em modo texto e gere as imagens com a ferramenta Graphviz mostrando a sequência de modificações das tabelas de roteamento para que o usuário as visualize.

3.3.1 Implementação de um programa para gerar a saída gráfica

Para gerar a saída gráfica do Observador Didático foi desenvolvido um programa que automaticamente lê os arquivos com as tabelas de roteamento salvas no diretório `resultados`, e gera as imagens em sequência para o usuário utilizando a ferramenta Graphviz. Para isso foi desenvolvido um programa utilizando a linguagem de programação Java e a plataforma de desenvolvimento NetBeans.

O programa precisa saber a topologia de rede que tem que ser gerada, o diretório onde estarão as tabelas de roteamento salvas e os roteadores que serão monitorados. Para isso o usuário deve informar ao programa a localização do arquivo `lab.conf` do laboratório do Observador Didático, já que nesse arquivo está descrito a topologia de rede, e sabendo a localização desse arquivo, sabe-se a localização onde estão salvas as tabelas de roteamento, que é o diretório `resultados` no mesmo diretório do arquivo `lab.conf`.

O arquivo `lab.conf` será lido e interpretado pelo programa, codificando o arquivo do Netkit que descreve a topologia de rede, para um objeto da linguagem Java que será criado chamado topologia, contendo todas as informações da topologia de rede que precisa gerar-se com a ferramenta Graphviz. Esse objeto topologia terá a descrição de todos os nós da rede com seus respectivos nomes e suas interfaces de rede. As interfaces de rede ainda terão a descrição de seus nomes e a quais interfaces de rede de outros nós da rede ela está conectado. Com o objeto topologia é possível saber toda a descrição da rede, com todas as suas conexões.

Com o programa sabendo a topologia de rede, será apresentado ao usuário uma tela de configuração com os nomes de todos os nós da rede, para ser configurado quais nós da rede serão computadores e quais serão roteadores, pra determinar qual figura será usada para representar aquele vértice com a ferramenta Graphviz. Os nós de rede configurados como roteadores, ainda podem ser configurados para ser mostrado as suas tabelas de roteamento, sabendo-se quais rodeadores serão monitorados.

Sabendo-se a topologia de rede e os roteadores que serão monitorados, pode-se mostrar ao usuário a imagem gerada com a ferramenta Graphviz, para isso precisa ser criado um arquivo texto na linguagem utilizado pela ferramenta Graphviz, que descreve o grafo que represente essa topologia de rede. Foi feito no programa uma classe responsável por receber o objeto topologia

e criar o arquivo corresponde que represente essa topologia de rede com a ferramenta Graphviz. Podendo ser visto a imagem que representa a topologia de rede pelo usuário, é possível mudar algumas poucas opções para tentar gerar imagens que representem melhor a topologia de rede, antes de gerar a sequência de imagens com as mudanças na tabela de roteamento, que terão o mesmo padrão de imagens.

Pode-se utilizar o programa de duas maneiras para visualizar as mudanças nas tabelas de roteamento, depois que usuário já está visualizando a imagem da topologia de rede. Uma delas é com o laboratório do Netkit em execução, ao mesmo tempo que as máquinas virtuais estão sendo executadas. A outra é com o Netkit parado, depois que já executou-se o laboratório e que já foram salvas as tabelas de roteamento com os resultados.

Para ambos os casos são criadas várias *Threads*, uma para cada arquivo que terá as tabelas de roteamento salvas, onde essas *Threads* serão responsáveis por ler esses arquivos e identificar as mudanças nas tabelas de roteamento, passando essas mudanças para uma lista que irá conter todas as mudanças de todas as tabelas de roteamento, com as informações de qual roteador é a tabela modificada e qual tempo a tabela mudou. Nessa lista as mudanças nas tabelas de roteamento serão organizados na ordem temporal que aconteceu as mudanças, sendo outra *Thread* responsável por pegar uma a uma essas mudanças e gerar a imagem com a ferramenta Graphviz, alterando somente o vértice que representa a tabela que foi modificada , através de um método da classe que gerou o arquivo para a ferramenta Graphviz. Foi implementado outros recursos e mecanismos que com as figuras já geradas irão mostrar isso ao usuário.

A utilização do programa implementado para o acompanhamento da evolução das tabelas de roteamento pode ser visto na Figura 3.8. O programa apresenta no canto superior esquerdo três botões que com eles é possível avançar/retroceder as figuras geradas e um modo de exibição automático, onde as figuras irão ser mostradas no tempo que são geradas, sendo exibidas por um certo período de tempo na tela. Do outro lado, no canto superior direito há um contador para se saber qual figura está sendo observada e o total de figuras geradas, assim pode-se saber quantas mudanças ocorreram nas tabelas de roteamento, já que cada figura é uma mudança. Também existe uma indicação do tempo, com a qual é possível saber o tempo passado deste a primeira mudança nas tabelas. A primeira mudança é considerado o tempo zero, e a indicação do lado o Δt indica o tempo decorrido entre duas mudanças seguidas. Outro recurso do programa que pode ser visualizado na figura é a exibição das interfaces que estão desativadas, sendo exibidos seus nomes em vermelho e o enlace que conecta essa interface aparece pontilhada, para poder visualizar que aquele enlace foi derrubado, como no caso o roteador Brasil e sua interface eth0.

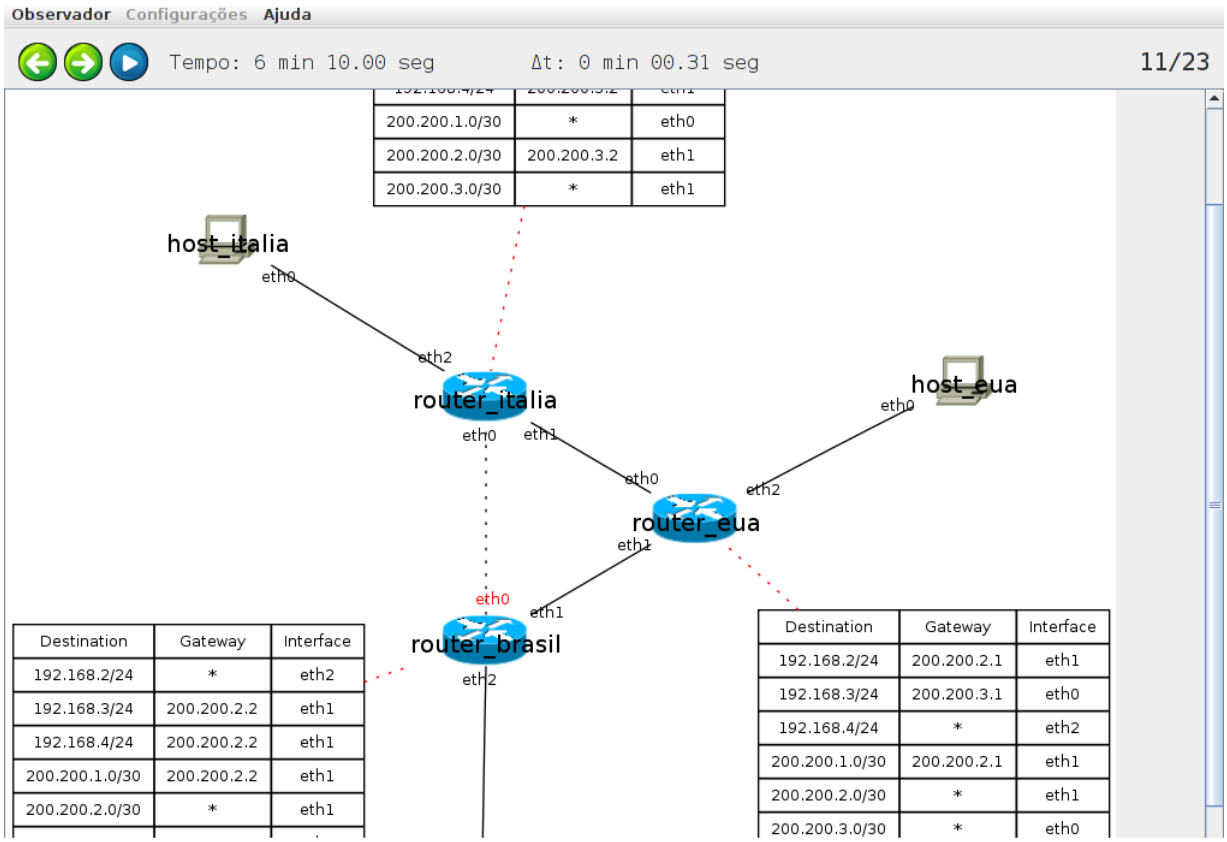


Figura 3.8: Observador Didático em funcionamento com a visualização em forma gráfica.

3.4 Extensão da ferramenta para observação de outros protocolos de roteamento

Com o Observador Didático funcionando juntamente com o programa para visualizar de forma gráfica as tabelas de roteamento foram realizados testes e a extensão da ferramenta para observar também outros protocolos de roteamento. Com o uso do Netkit e as suas facilidades, pode-se facilmente mudar o protocolo do roteamento sem interferir no funcionamento do Observador Didático, conseguindo observar as mudanças utilizando outros protocolos de roteamento.

Para observador outro protocolo de roteamento, apenas é preciso fazer as suas configurações, criando os arquivos necessários no diretório da máquina virtual que será o roteador da rede, e iniciar os serviços referentes ao protocolo de roteamento, que o Observador Didático funcionará corretamente capturando as tabelas de roteamento, sem a necessidade de grandes mudanças e alterações no laboratório do Netkit.

4 *Resultados e testes*

Neste capítulo será descrito os resultados obtidos com o Observador Didático após a implementação das melhorias, descritas no capítulo 3. Para isso será mostrado os resultados obtidos executando-se um laboratório utilizando o Observador Didático, e depois os resultados obtidos usando outro protocolo de roteamento no mesmo laboratório, comparando os seus resultados. No final do capitulo será descrito as limitações que foram encontrados com a implementação realizada do Observador Didático.

4.1 Executando um Laboratório com o Observador Didático

Para mostrar os resultados obtidos e o funcionamento do Observador Didático será executado um laboratório utilizando o Netkit, para salvar as tabelas de roteamento e depois visualizá-las com o programa que implementa a saída gráfica. Esse laboratório utilizado é o que foi implementado no capítulo 3, e utilizado o protocolo de roteamento RIP nos seus roteadores. A execução do laboratório consistiu de iniciar todas as máquinas virtuais e deixar estabilizar o protocolo de roteamento, depois foi derrubada a interface eth0 do roteador Brasil, esperou-se de novo a estabilização do protocolo de roteamento, e por último foi reerguida a interface eth0 do roteador Brasil.

Foi utilizado o programa que implementa a saída gráfica usando o método de execução com o Netkit parado, já que as informações já estão salvas. Na Figura 4.1, é possível ver a primeira tabela de roteamento capturada, que é do roteador EUA, logo após ele iniciar. No programa aparece que é a segunda imagem gerada, pois a primeira é a imagem para mostrar a topologia de rede com o Graphviz. Nessa tabela foram adicionadas automaticamente as rotas das redes a qual o roteador EUA está diretamente conectado, que é um comportamento normal dos dispositivos IP. As tabelas dois e três capturadas representam o mesmo ocorrido com o roteador EUA, que são as rotas automaticamente adicionadas aos roteadores Brasil e Itália logo após serem inicializados, como pode ser visto na Figura 4.2. Pela indicação do tempo, percebe-se que as máquinas virtuais iniciaram ao mesmo tempo, passando-se somente um segundo entre

as tabelas capturadas.

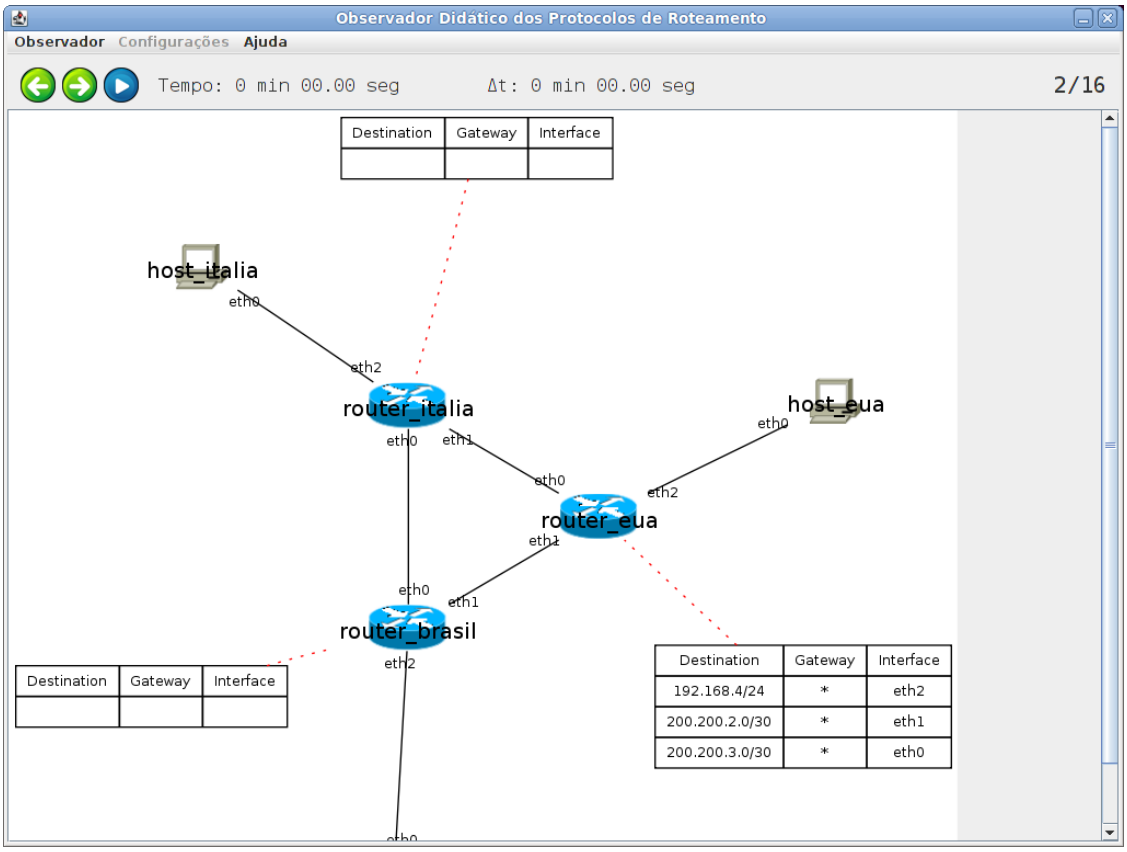


Figura 4.1: Tabela de roteamento capturada - 01 - RIP

Logo após a inicialização das máquinas virtuais, e de serem capturadas as primeiras tabelas de roteamento de cada roteador, o protocolo de roteamento começará seu funcionamento. Na Figura 4.3, aproximadamente sete segundos depois da inicialização de todas as máquinas virtuais, o roteador Brasil adicionou à sua tabela de roteamento, as rotas das redes que não estão diretamente conectados a ele. Essas redes são a do computador EUA, do computador Itália e da rede que conecta os roteadores EUA e Itália. Nesses sete segundos o roteador Brasil recebeu dos roteadores EUA e Itália mensagens com as redes que eles conheciam, e com essa informação adicionou as rotas para essas redes. O mesmo ocorreu com com os roteadores EUA e Itália, menos de meio segundo depois, adicionando todas as rotas para as redes que não se conhecia de uma única vez. Na Figura 4.4, é possível ver o protocolo de roteamento estabilizado, com os roteadores conhecendo todas as rotas possíveis, isso levou aproximadamente dez segundos após a captura da primeira tabela de roteamento.

Essa sequência de tabelas capturadas, podem sofrer variações a cada vez que o laboratório é executado novamente, sendo capturadas mais tabelas ou menos. Algumas vezes, um roteador aprende todas as rotas para as redes disponíveis com duas mudanças na tabela de roteamento. Por exemplo, o roteador Brasil em uma mudança adiciona as rotas aprendidas pelo roteador

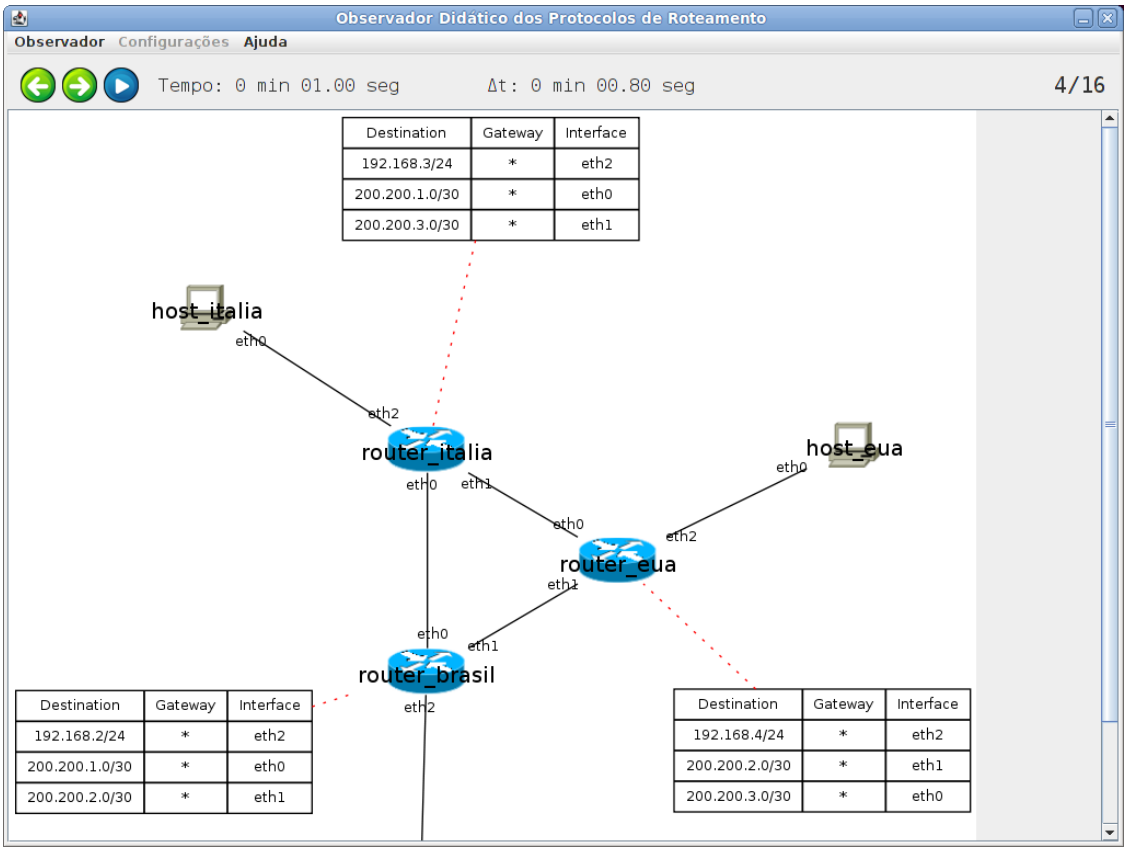


Figura 4.2: Tabela de roteamento capturada - 03 - RIP

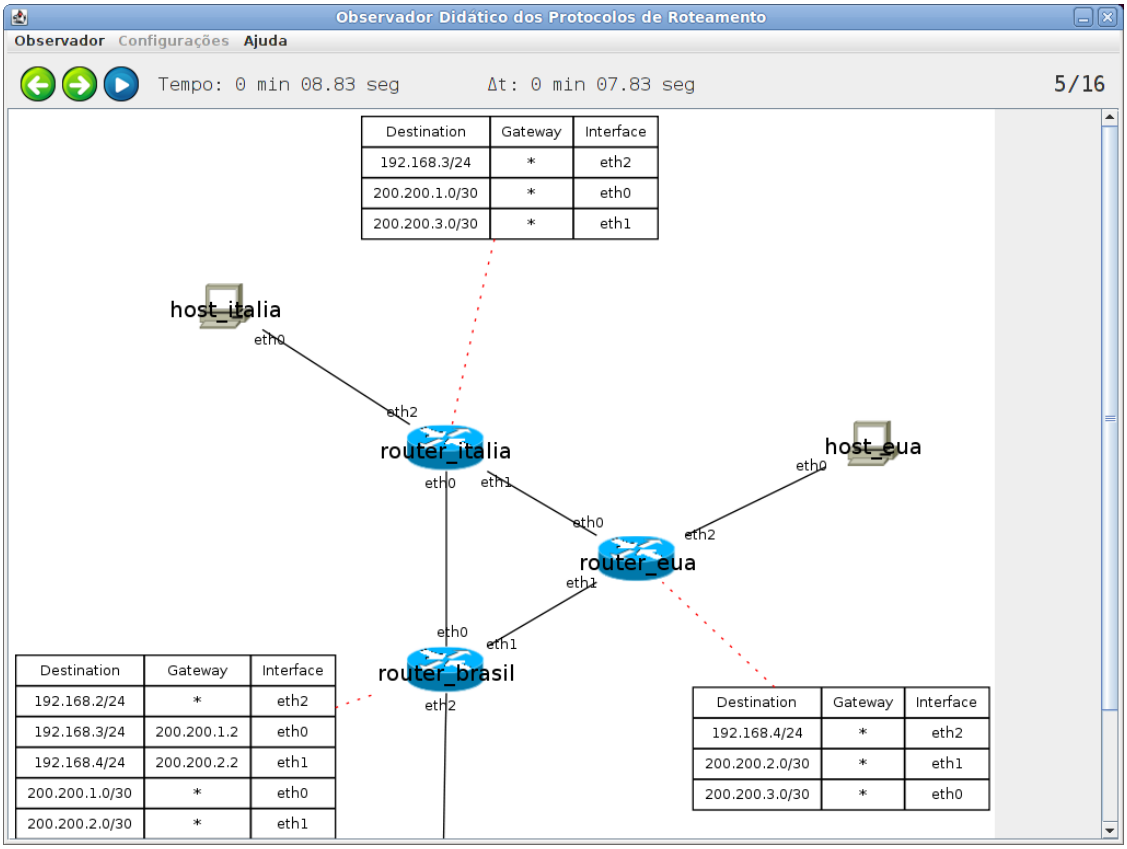


Figura 4.3: Tabela de roteamento capturada - 04 - RIP

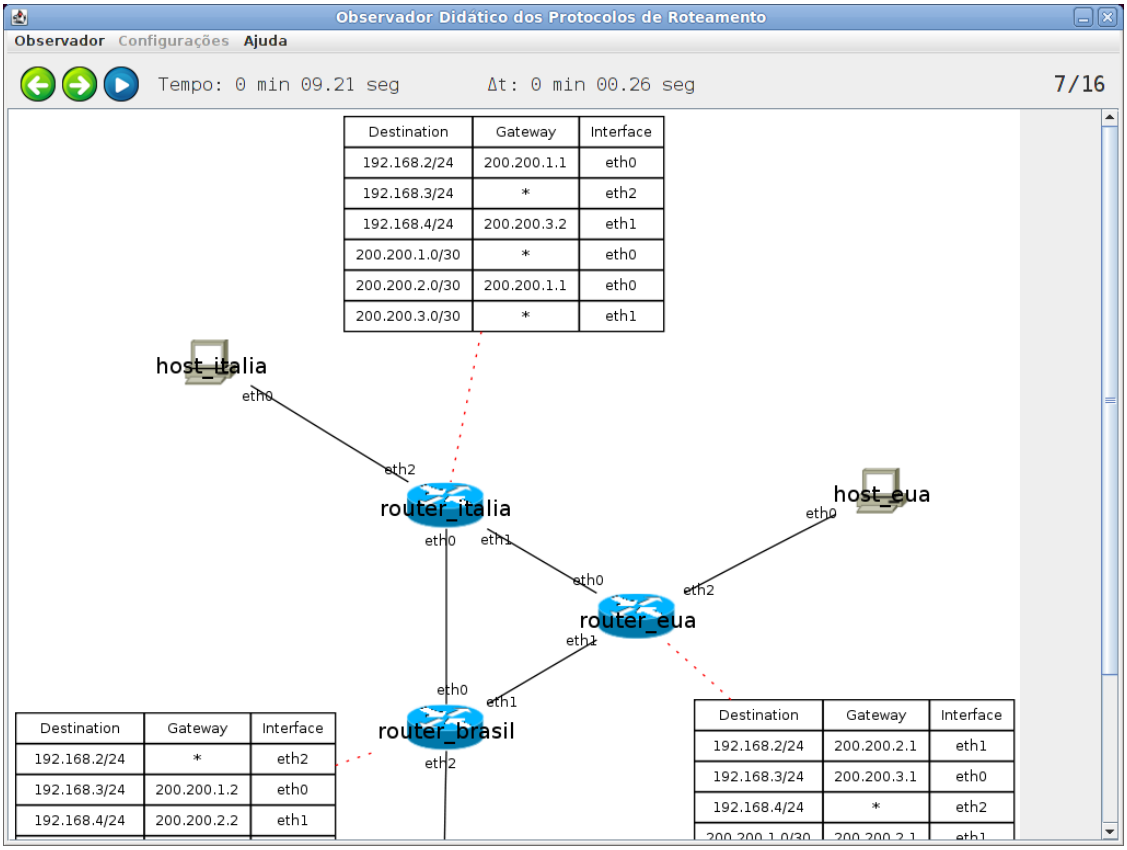


Figura 4.4: Tabela de roteamento capturada - 06 - RIP

Itália, faltando a rota para o computador EUA, em seguida ocorre uma nova mudança o roteador Brasil adiciona a rota para o computador EUA. Outra sequência que ocorre, por exemplo, o roteador Brasil aprender todas as rotas. A rota para a rede que conecta os roteadores EUA e Itália é feita através do roteador Itália, e em uma próxima mudança, o roteador Brasil troca essa rota pela rota indo pelo roteador EUA. Essas mudanças ocorrem muito rapidamente, em poucos segundos.

Agora veremos o que ocorre quanto o enlace eth0 do roteador Brasil é derrubado depois de estabilizado as tabelas de roteamento. Na Figura 4.5 é possível ver o que aconteceu. O roteador Brasil ao ter seu enlace eth0 derrubado, perdeu todas as todas rotas que utilizam essa interface de rede, não tendo mais rotas para essas redes. É possível visualizar que na imagem é representado que o a interface foi derrubada, aparecendo seu nome em vermelho, e o enlace que utiliza essa interface aparece pontilhado.

Aproximadamente nos dezessete segundos seguidos, ocorrem quatro mudanças nas tabelas de roteamento dos roteadores Brasil e EUA, sendo duas mudanças para cada. O roteador EUA em seguida, depois que o roteador Brasil perdeu suas rotas que utilizam a interface de rede eth0, perde a rota que utilizava o roteador Brasil para chegar no enlace Brasil-Itália, ficando sem rota

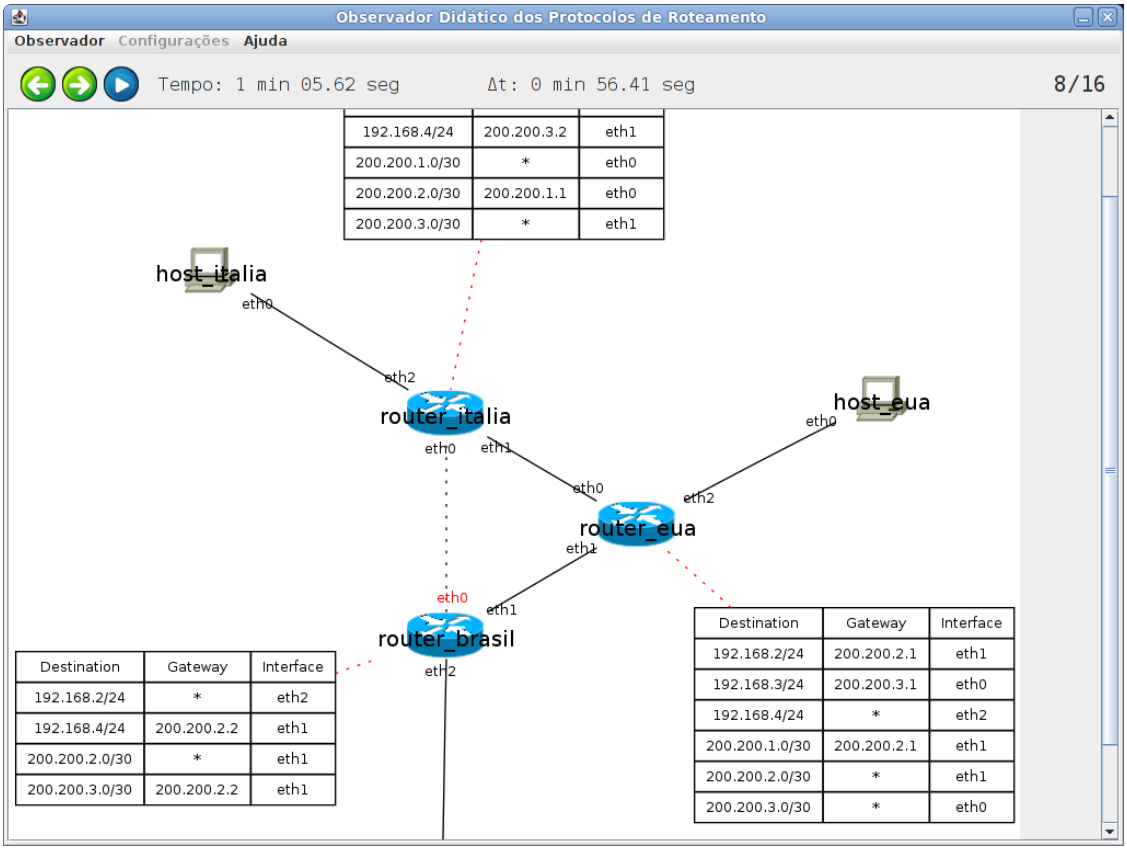
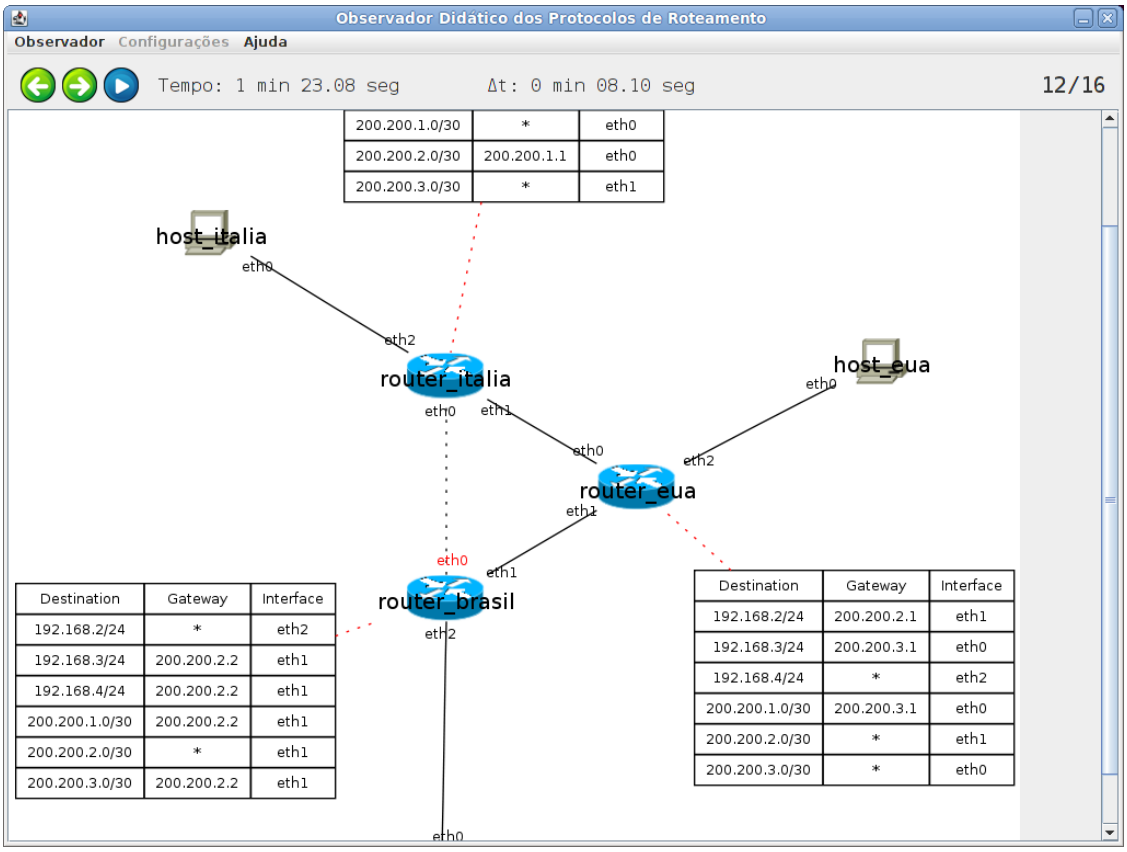


Figura 4.5: Tabela de roteamento capturada - 07 - RIP

para essa rede. Em sequência, o roteador Brasil adiciona uma nova rota através do roteador EUA para voltar á ter acesso ao enlace Brasil-Itália. Continuando a sequência, o roteador EUA adiciona uma nova rota através do roteador Itália para recuperar o acesso ao enlace Brasil-Itália que havia sido perdido. A última mudanças das quatro que ocorreram, o roteador Brasil adiciona uma nova rota através do roteador EUA, para ter acesso ao computador Itália, recuperando assim suas rotas perdidas. As novas tabelas de roteamento dos roteadores Brasil e EUA estabilizadas podem ser vista na Figura 4.6, nota-se que o roteador Brasil utiliza a interface eth1 em praticamente todas as suas rotas agora.

O roteador Itália utiliza o enlace Brasil-Itália como rota para chegar ao computador Brasil. Mas com a interface eth0 do roteador Brasil derrubada, o roteador Itália não pode mais usar essa rota. Somente após 2 minutos e 20 segundos da última mudança ter acontecido com o roteador Brasil, que o roteador Itália muda sua rota para o computador Brasil, invés de utilizar o roteador Brasil, mudou para o roteador EUA. Nesse tempo o roteador Itália perdeu a comunicação com o computador Brasil. Isso pode ser visto na Figura 4.7, repara-se o tempo passado entre a última modificação e a atual.

Com todos os roteadores com as suas tabelas de roteamento estabilizadas, após a queda da

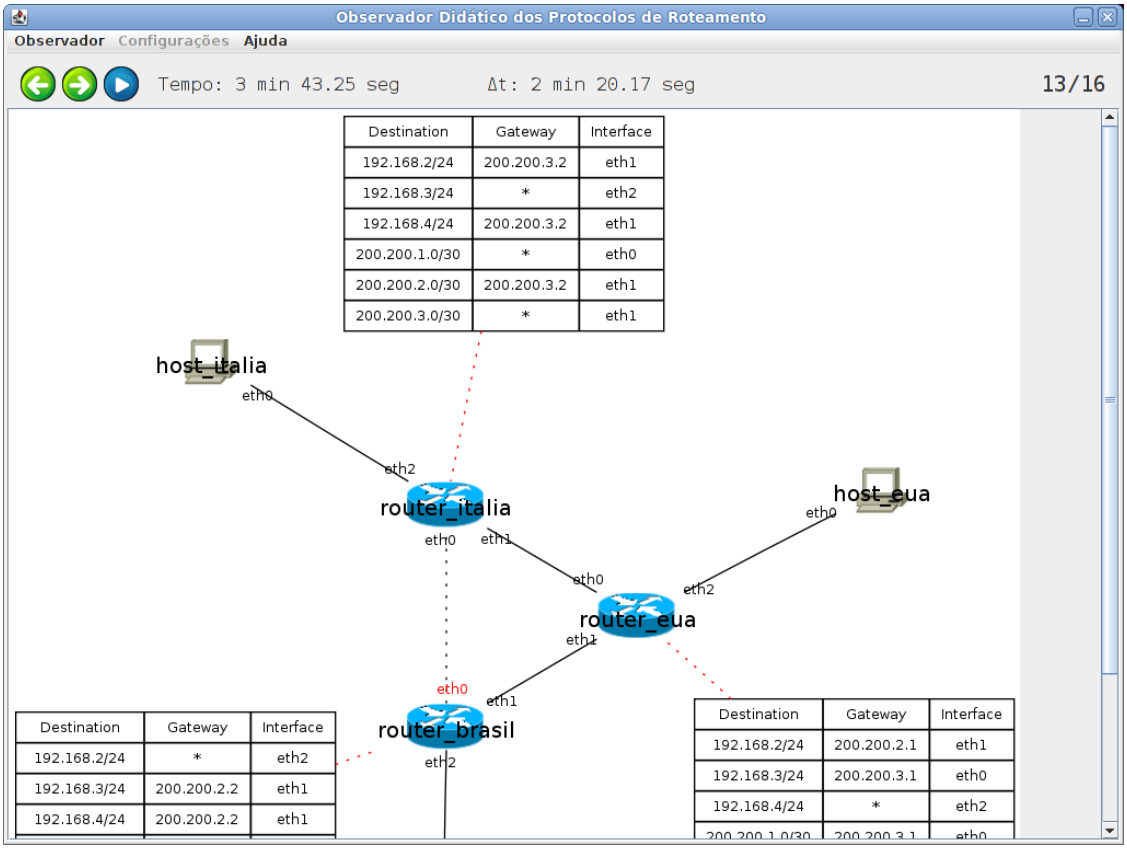


200.200.1.0/30	*	eth0
200.200.2.0/30	200.200.1.1	eth0
200.200.3.0/30	*	eth1

Destination	Gateway	Interface
192.168.2/24	*	eth2
192.168.3/24	200.200.2.2	eth1
192.168.4/24	200.200.2.2	eth1
200.200.1.0/30	200.200.2.2	eth1
200.200.2.0/30	*	eth1
200.200.3.0/30	200.200.2.2	eth1

Destination	Gateway	Interface
192.168.2/24	200.200.2.1	eth1
192.168.3/24	200.200.3.1	eth0
192.168.4/24	*	eth2
200.200.1.0/30	200.200.3.1	eth0
200.200.2.0/30	*	eth1
200.200.3.0/30	*	eth0

Figura 4.6: Tabela de roteamento capturada - 11 - RIP



Destination	Gateway	Interface
192.168.2/24	200.200.3.2	eth1
192.168.3/24	*	eth2
192.168.4/24	200.200.3.2	eth1
200.200.1.0/30	*	eth0
200.200.2.0/30	200.200.3.2	eth1
200.200.3.0/30	*	eth1

Destination	Gateway	Interface
192.168.2/24	*	eth2
192.168.3/24	200.200.2.2	eth1
192.168.4/24	200.200.2.2	eth1

Destination	Gateway	Interface
192.168.2/24	200.200.2.1	eth1
192.168.3/24	200.200.3.1	eth0
192.168.4/24	*	eth2

Figura 4.7: Tabela de roteamento capturada - 12 - RIP

interface eth0 do roteador Brasil, a interface de rede eth0 do roteador Brasil foi reerguida. Isso causou três mudanças nas tabelas de roteamento dos roteadores Brasil e Itália. Percebe-se que não houve mais modificações nas tabelas de roteamento do roteador EUA, ficando-se com a rota pelo roteador Itália para o enlace Brasil-Itália, que no início era pelo roteador Brasil. A primeira mudança dessas três, ocorreu na tabela de roteamento no roteador Brasil, quanto a interface eth0 foi reerguida, a mudança pode ser vista na Figura 4.8. A mudança que aconteceu foi que a rota para o enlace Brasil-Itália que era feita pelo roteador EUA foi perdida, no lugar colocou-se a rota utilizando diretamente a interface eth0, que está conectada diretamente no enlace Brasil-Itália. A próxima mudança também ocorre no roteador Brasil, menos de um segundo depois da última, onde é adicionada a rota que tinha sido perdido para o computador Itália através do roteador Itália, removendo-se a rota través do roteador EUA. Volta-se a utilizar essa rota para o computador Itália através do roteador Itália, porque é a rota que apresenta o menor custo. Como última mudança, o roteador Itália volta utilizar a rota pelo roteador Brasil, para chegar no computador Brasil, removendo a rota que ia pelo roteador EUA.

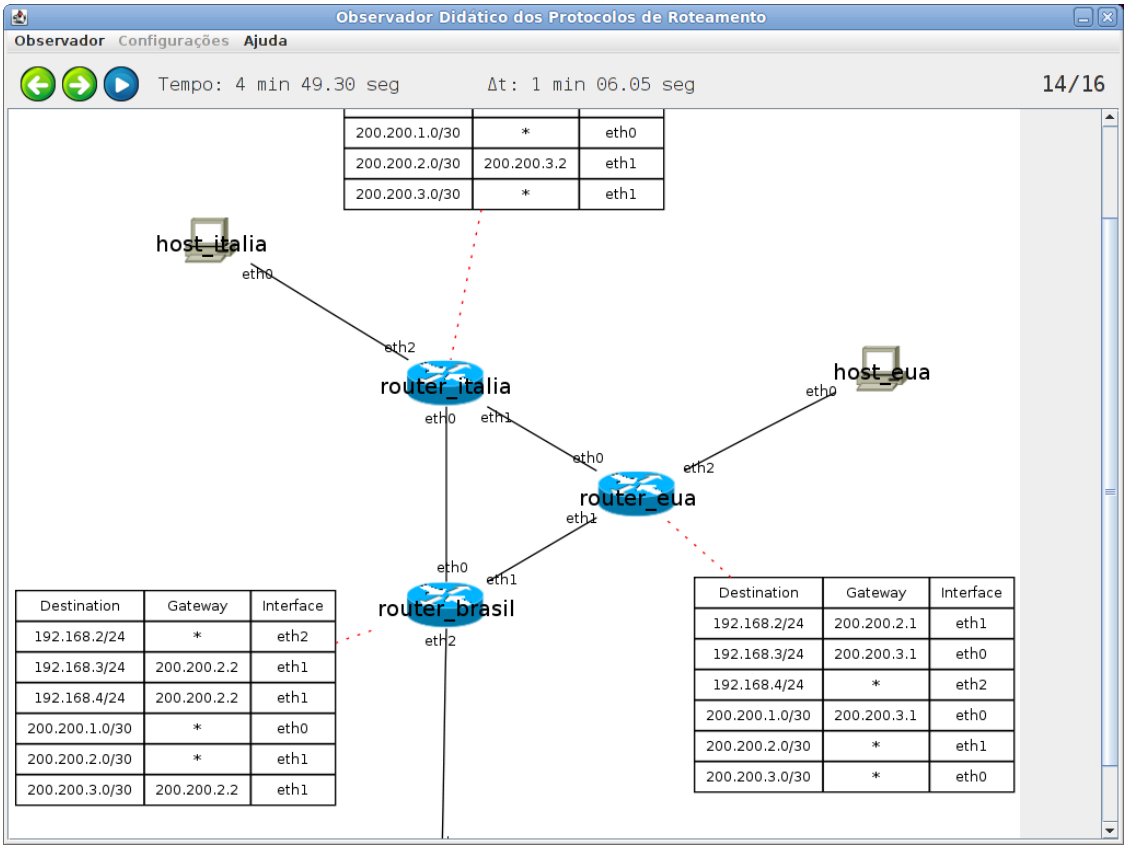


Figura 4.8: Tabela de roteamento capturada - 13 - RIP

4.2 Executando um Laboratório com o Observador Didático utilizando outro protocolo de roteamento

Será mostrado agora o mesmo laboratório executado na secção 4.1, mas utilizando-se outro protocolo de roteamento, para comparar-se com os resultados anteriores. O protocolo de roteamento escolhido para substituir o RIP é o *Open Shortest Path First* (OSPF). Iremos ver apenas as diferenças que ocorrem na evolução das tabelas de roteamento, para discuti-las, e não todo o processo que ocorreu.

A evolução das tabelas de roteamento inicialmente são iguais para ambos os protocolos de roteamento, sendo capturadas as tabelas iniciais de cada roteador, e em seguida as tabelas com as rotas para as redes existentes adicionadas. O que diferencia-se é o tempo necessário para isso ocorrer, para as tabelas de roteamento se estabilizarem, como pode ser visto na Figura 4.9, onde as tabelas se estabilizaram aproximadamente 50 segundos depois.

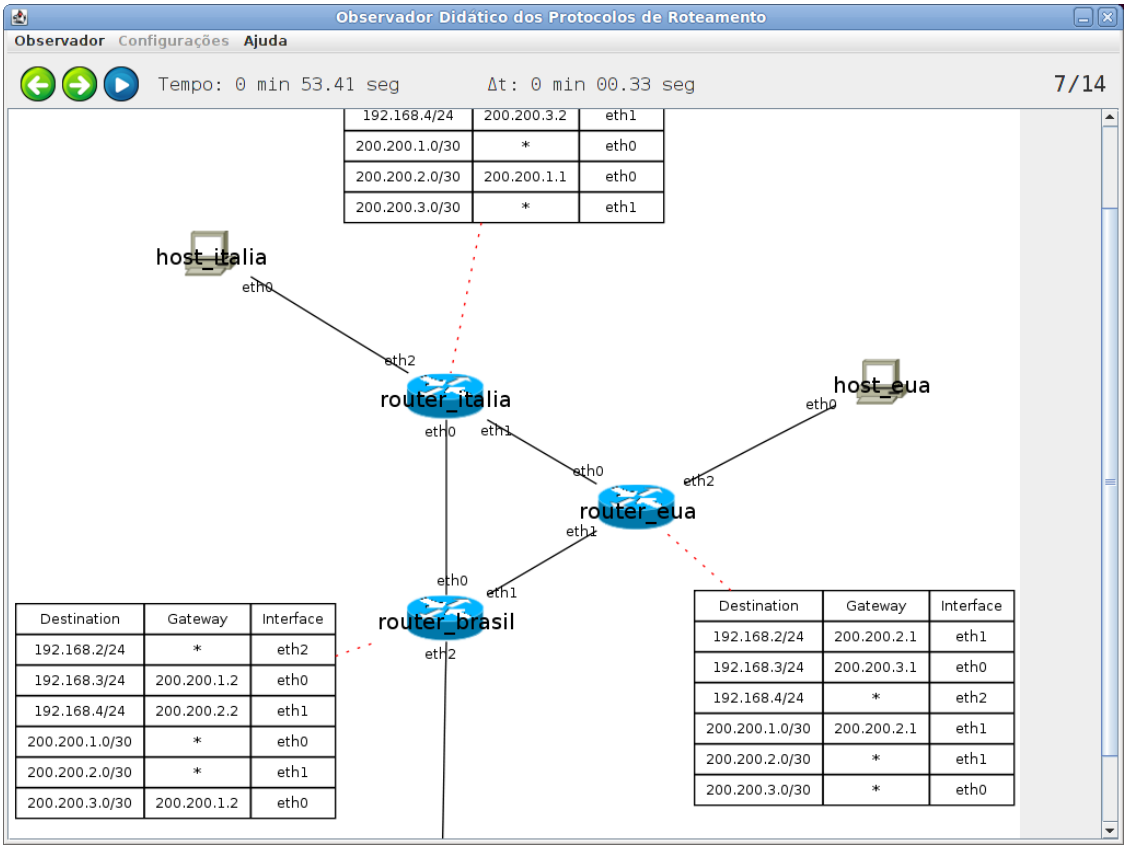


Figura 4.9: Tabela de roteamento capturada - 06 - OSPF

A grande diferença que encontra-se entre os protocolos é quando a interface eth0 do roteador Brasil é derrubada. Como pode ser visto na Figura 4.10, que é a primeira tabela de roteamento capturada depois que a interface foi derrubada. Percebe-se que o roteador Brasil removeu as rotas que utilizam a interface de rede eth0 que foi derrubada, e colocou novas rotas através

do roteador EUA para substituir as que foram removidas. Estabilizando sua tabela de roteamento mais rapidamente e em apenas uma única mudança capturada. O mesmo ocorre com os roteadores EUA e Itália, que em menos de um segundo depois, estabilizam suas tabelas de roteamento, ocorrendo apenas duas mudanças. Primeiro o roteador EUA, em apenas uma mudança capturada, que utilizava antes o roteador Brasil para chegar ao enlace Brasil-Itália, removeu essa rota e a substituiu por uma rota utilizando o roteador Itália. Com o roteador Itália utilizando o protocolo de roteamento RIP, levou-se pouco mais de 2 minutos para corrigir suas rotas. Utilizando o protocolo de roteamento OSPF, em um segundo apenas o roteador Itália corrigiu suas rotas, removendo as que utilizam o roteador Brasil, e adicionando as rotas utilizando o roteador EUA, como pode ser visto na Figura 4.11.

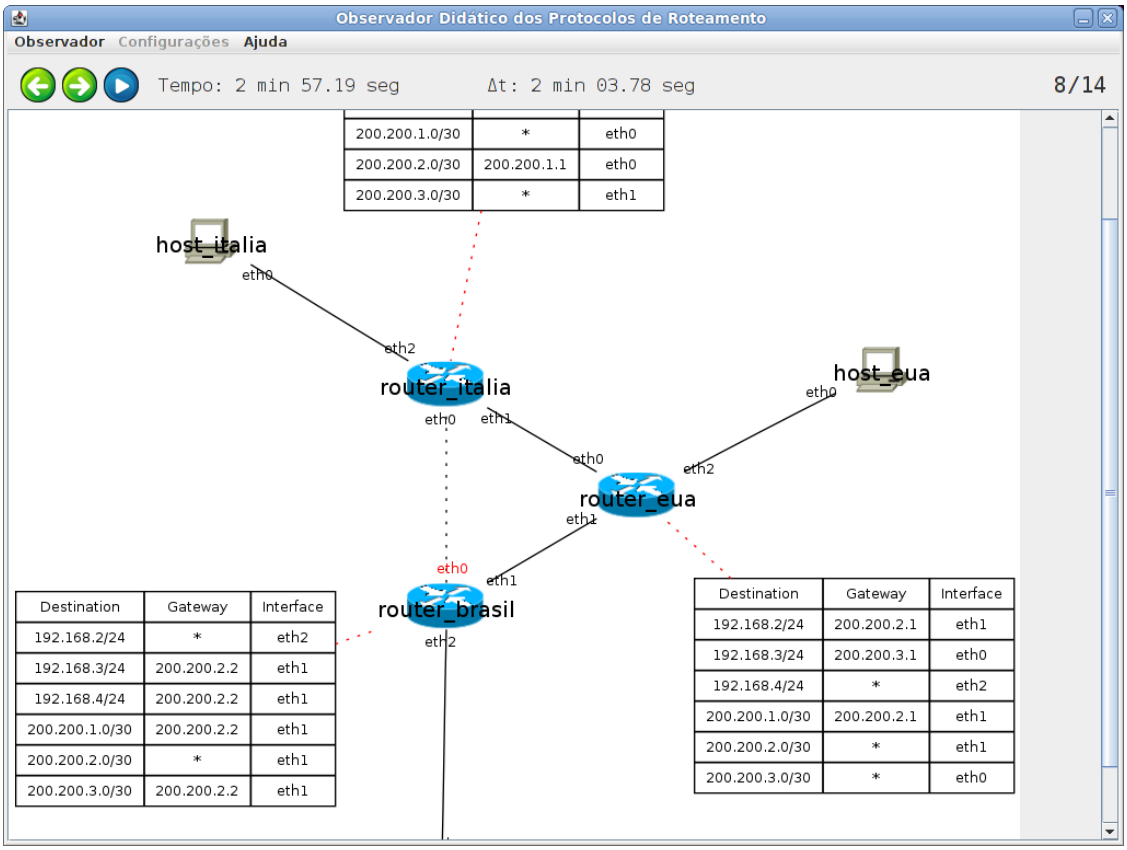


Figura 4.10: Tabela de roteamento capturada - 07 - OSPF

Percebe-se que o protocolo OSPF atua mais rapidamente para recompor as tabelas de roteamento quanto um enlace é derrubado, trocando as rotas perdidas por outras quase instantaneamente. Uma das possíveis causas para isso ocorrer, é que o OSPF utiliza o algoritmo de roteamento de estado de enlace, sendo um protocolo de roteamento global, tendo conhecimento completo e total da rede, diferente do RIP que utiliza o algoritmo de roteamento vetor de distâncias, sendo um protocolo de roteamento descentralizado, que só conhece as informações recebidas de seus vizinhos. Segundo Kurose e Ross (2006) o algoritmo vetor de distâncias pode convergir

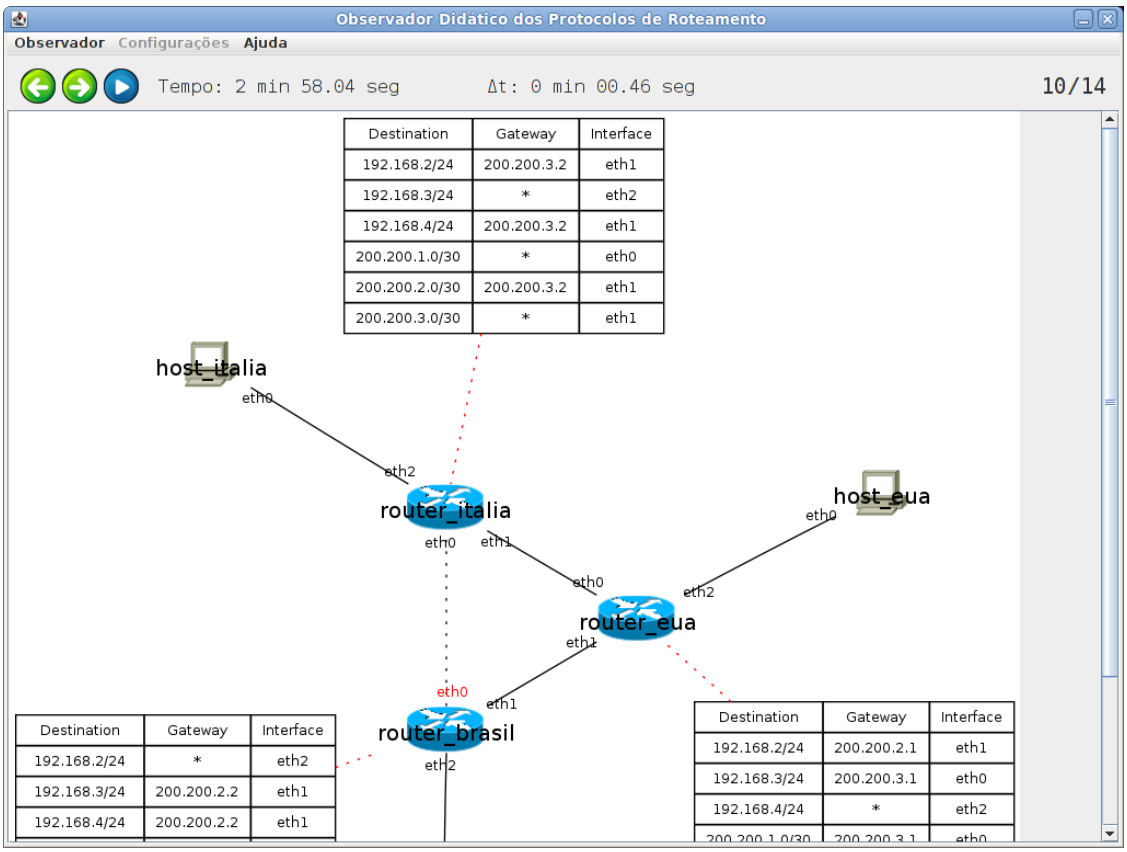


Figura 4.11: Tabela de roteamento capturada - 09 - OSPF

mais lentamente e ter *loops* de roteamento enquanto estiver convergindo. O protocolo de roteamento OSPF sabendo a topologia completa da rede, ao se derrubar um enlace, consegue atuar rapidamente para repor a rota perdida se disponível.

O mesmo ocorrido quanto a interface eth0 do roteador Brasil foi derrubado, aconteceu quanto ela foi reerguida, utilizando-se o protocolo de roteamento OSPF. As tabelas de roteamento foram rapidamente estabilizadas, voltando-se as rotas que se tinha antes da interface ser derrubada.

4.3 Limitações encontradas com a implementação realizada

Como mostrado nesse capítulo, o Observador Didático com interface gráfica funciona corretamente para se acompanhar as tabelas de roteamento dos roteadores. Contudo ao executar-se inúmeras vezes os laboratórios, e com uma análise mais detalhada, foram encontrados limitações no Observador Didático. Mas essas limitações não o impossibilitam de ser usado para o seu propósito.

Uma dessas limitações é quanto a ferramenta Graphviz gera uma imagem de resolução

muito alta, na ordem de mais de 5000 *pixels*. Quanto essa imagem for ser exibida ao usuário, resultará em um estouro de memória da *Java Virtual Machine* (JVM), causando erro e extrema lentidão no programa. Só acontece isso em se gerar imagens de uma grande estrutura de rede, principalmente utilizando o programa `circo` do Graphviz. Podem ser geradas imagens com muito mais máquinas do que as apresentadas nesse capítulo, só tomando o cuidado para não gerar imagens grandes demais com o programa `circo` do Graphviz.

Nesse capítulo foi mostrado a utilização do Observador Didático com o Netkit parado, mas poderia ser utilizado com o Netkit em execução também, se obtendo o mesmo resultado. Poucas vezes utilizando-se o Observador Didático com o Netkit em execução, ocorreu que duas modificações seguidas das tabelas de roteamento são trocadas suas ordens. Por exemplo, uma modificação B que ocorre depois de outra modificação A, a modificação B será exibida antes da modificação A, sendo possível perceber isso, pois na informação de tempo decorrido entre duas mudanças seguidas aparecerá um número negativo.

O Observador Didático ainda não é capaz de capturar todas as mudanças que ocorrem na tabela de roteamento de um roteador, mudanças seguidas com diferença de tempo muito curto que ocorrem em um mesmo roteador, ainda são capturadas repetidas. Mas existe variação a cada execução do Observador Didático, algumas vezes chega há não ter tabelas repetidas nos arquivos textos onde elas ficam salvas.

Além das limitações acima, também foram encontradas outras. Que trabalhando-se nelas puderam ser resolvidas. As limitações descritas acima são as que não conseguiu-se resolver, mas há chance de algumas serem resolvidas, continuando-se trabalhando nelas.

5 *Conclusões*

Este trabalho teve como objetivo a implementação de várias melhorias no Observador Didático com intuito de aperfeiçoa-lo. Com as melhorias descritas para serem implementadas, espera-se que o Observador Didático se torne uma ferramenta para ser usada no ensino dos protocolos de roteamento, um auxílio para ajudar no entendimento.

Os objetivos descritos no capítulo 1 foram todos atingidos ao decorrer desse trabalho. Foi conseguido realizar a implementação de todas as melhorias pretendidas. Com essas melhorias, o Observador Didático tornou-se uma ferramenta capaz de ser usada para o ensino dos protocolos de roteamento, em grande parte devido ao programa implementado para mostrar de forma gráfica as mudanças nas tabelas de roteamento. Esse programa é de fácil uso para o usuário e apresenta um bom resultado para o acompanhamento das tabelas de roteamento.

A utilização do protocolo SNMP na construção do Observador Didático para acompanhar as mudanças nas tabelas de roteamento foi o mecanismo escolhido por Farias (2010). Na implementação das melhorias continuou-se usando esse mecanismo, e não pensou-se em sua alteração. Se o Observador Didático fosse de uso geral, para ser usado em redes reais, o uso do protocolo SNMP se justificaria, mas a ferramenta Observador Didático é de uso restrito ao Linux, utilizando a ferramenta Netkit, e nesse caso existem outros mecanismos mais eficientes.

O Observador Didático com a interface gráfica é uma opção para acompanhar as mudanças nas tabelas de roteamento que são causadas pelos protocolos de roteamento, servindo como ajuda para entendê-los. Para se entender completamente o funcionamento dos protocolos de roteamento, seria necessário capturar todas as mensagens trocada entre os protocolos de roteamentos e o número de mensagens que são necessárias para causar uma mudança na tabela de roteamento.

ANEXO A – Exemplo de uso das máquinas virtuais UML

Na Figura A.1, pode ser vista a rede que será usada como exemplo, para o uso das máquinas virtuais UML. A rede é composta por quatro computadores e dois roteadores que fazem a ligação entre os computadores.

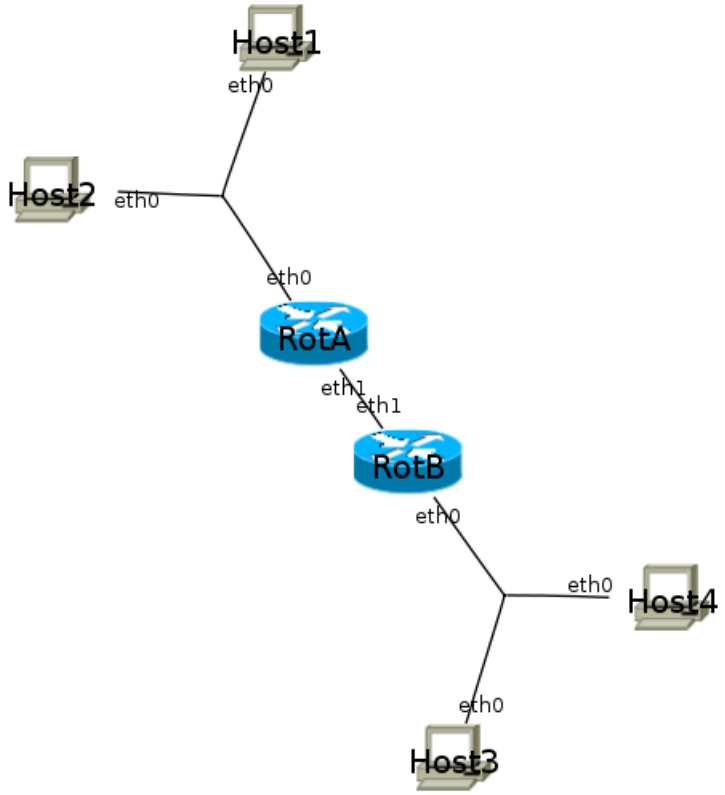


Figura A.1: Topologia de rede usada para o exemplo.

Para poder-se criar a rede da Figura A.1 utilizando as máquinas virtuais UML, precisa ser criado um `script` que inicialize essa rede. Esse *script* irá lançar os UML Switches necessários para fazer todas as conexões entre as máquinas virtuais, que nesse caso serão três. Depois é preciso inicializar todas as máquinas virtuais utilizando o UML Kernel com todos os parâmetros necessários. Esses parâmetros seriam as opções para definir os nomes das máquinas virtuais, o

sistema de arquivos que será utilizado juntamente com seu arquivo COW, e conectar as interfaces de rede aos UML Switches lançados. Também é preciso iniciar cada máquina virtual em um terminal separado usando o Xterm. Esse `script` pode ser visto na Figura A.2.

```
#!/bin/bash

#lancando os UML switches necessarios
uml_switch -unix /tmp/switch1.ctl &
uml_switch -unix /tmp/switch2.ct2 &
uml_switch -unix /tmp/switch3.ct2 &

#iniciando os roteadores
xterm -geometry 60x20 -T RotA -e "./uml_kernel umid=RotA ubda=./cow-RotA,./
    Ubuntu mem=64M eth0=daemon,,,/tmp/switch1.ctl eth1=daemon,,,/tmp/switch2.ctl"
     &

xterm -geometry 60x20 -T RotB -e "./uml_kernel umid=RotB ubda=./cow-RotB,./
    Ubuntu mem=64M eth0=daemon,,,/tmp/switch3.ctl eth1=daemon,,,/tmp/switch2.ctl"
     &

#iniciando os computadores
xterm -geometry 60x20 -T Host1 -e "./uml_kernel umid=Host1 ubda=./cow-Host1,./
    Ubuntu mem=32M eth0=daemon,,,/tmp/switch1.ctl" &

xterm -geometry 60x20 -T Host2 -e "./uml_kernel umid=Host2 ubda=./cow-Host2,./
    Ubuntu mem=32M eth0=daemon,,,/tmp/switch1.ctl" &

xterm -geometry 60x20 -T Host3 -e "./uml_kernel umid=Host3 ubda=./cow-Host3,./
    Ubuntu mem=32M eth0=daemon,,,/tmp/switch3.ctl" &

xterm -geometry 60x20 -T Host4 -e "./uml_kernel umid=Host4 ubda=./cow-Host4,./
    Ubuntu mem=32M eth0=daemon,,,/tmp/switch3.ctl" &
```

Figura A.2: *Script* que inicializa a rede utilizada no exemplo.

Para poder se executar o `script` mostrado na Figura A.2 é necessário que ele esteja no mesmo diretório onde estão o arquivo `uml_kernel` e o arquivo do sistema de arquivo com nome `Ubuntu`. É preciso também que se tenha instalado a ferramenta UML Utilities, para poder se lançar os UML Switches. Ao executar esse `script` será inicializada as seis máquinas virtuais, cada uma em seu terminal, e será feito as conexões com os UML Switches. Mas as máquinas virtuais ainda não estarão totalmente preparadas, será preciso ir em cada máquina virtual e configurar manualmente suas interfaces de rede e endereços IP. Para encerrar as máquinas virtuais, elas podem ser desligadas normalmente ou utilizar o comando `killall uml_kernel` no terminal de comando, e para encerrar os UML Switches pode ser utilizado o mesmo comando, sendo `killall uml_switch`.

ANEXO B – Preparando a ferramenta Netkit para ser usada e exemplo de uso

B.1 Preparando Netkit para ser usado

Para poder-se usar o Netkit precisa ser baixado as três partes de que ele é composto, que podem ser obtidas no site do projeto do Netkit `http://www.netkit.org/`. Sendo que o Netkit funciona corretamente em muitas distribuições Linux, com uma lista de compatibilidade no site do projeto. As três partes são:

- netkit-X.Y.tar.bz2
- netkit-filesystem-FX.Y.tar.bz2
- netkit-kernel-KX.Y.tar.bz2

Onde as letras X e Y representarão as versões disponíveis. Depois de ter baixado todos os arquivos, deverão ser descompactados todos no mesmo diretório, formando um diretório chamado Netkit, com o conteúdo dos três arquivos descomprimidos.

Como ultimo passo deve-se configurar as variáveis de ambiente para poder se executar os comandos do Netkit, isso pode ser feito editando o arquivo `.profile` (que contém suas configurações do shell) na pasta pessoal do usuário, acrescentando as linhas mostradas na Figura B.1, considerando que o Netkit foi extraído na pasta pessoal do usuário.

```
1 export NETKIT_HOME=/home/seu-usuario/netkit
2 export MANPATH=:$NETKIT_HOME/man
3 export PATH=$NETKIT_HOME/bin:$PATH
```

Figura B.1: Configurando as variáveis de ambiente para executar os comandos do Netkit.

Depois de se entrar novamente com seu usuário, para carregar as novas configurações, pode-se testar se a configuração foi feita corretamente entrando no diretório do Netkit e executando

o *script* `check_configuration.sh`, se todos os testes feitos retornarem que foram bem sucedidos, o Netkit está pronto para ser usado. Lembrando que não é preciso ser usuário *root* para instalar o Netkit, nem para executá-lo.

B.2 Exemplo de uso da ferramenta Netkit

Vamos usar como exemplo a rede da Figura B.2 para mostrar o uso da ferramenta Netkit. A rede é composta por seis máquinas, sendo quatro computadores conectados uns aos outros por dois roteadores.

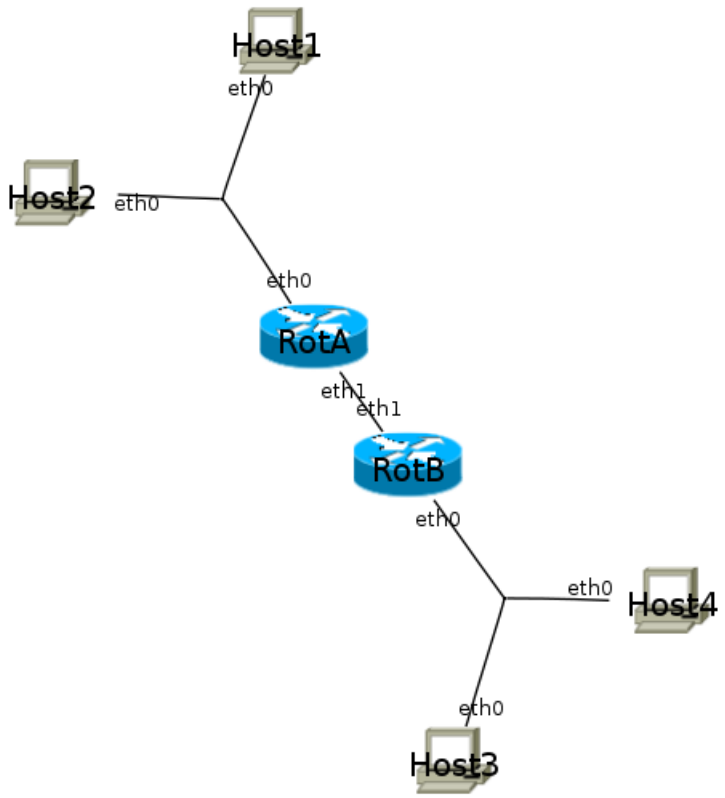


Figura B.2: Topologia de rede usada para o exemplo.

Primeiramente vamos criar um diretório para ser o laboratório do Netkit, dentro desse diretório iremos criar o arquivo `lab.conf`, onde será descrito a topologia de rede. No arquivo `lab.conf` iremos colocar os nomes das máquinas virtuais seguidas da sua interface e o domínio de colisão há qual elas estão conectadas. O arquivo `lab.conf` do exemplo pode ser visto na Figura B.3.

Dentro do diretório do laboratório é preciso criar um sub-diretório para cada máquina virtual com o mesmo nome usado no arquivo `lab.conf`. Deve-se criar também um arquivo `.startup` para cada máquina virtual com o seu nome na frente. A Figura B.4 mostra como

```
Host1[0]=DC1
Host2[0]=DC1

RotA[0]=DC1
RotA[1]=DC2

RotB[0]=DC3
RotB[1]=DC2

Host3[0]=DC3
Host4[0]=DC3
```

Figura B.3: Arquivo `lab.conf` da rede utilizada no exemplo.

deve ficar o diretório do laboratório do Netkit.

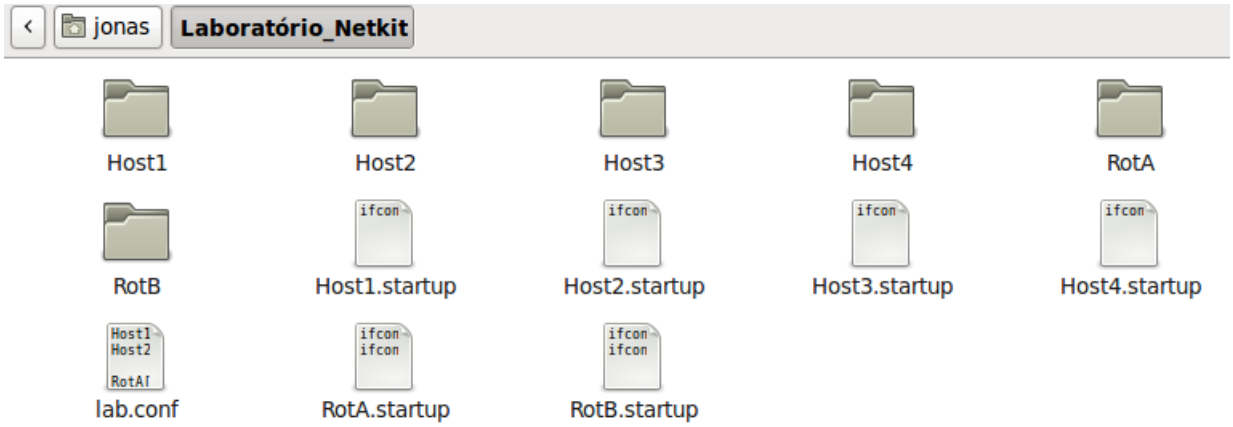


Figura B.4: Diretório do laboratório do Netkit.

Os arquivo `.startup` podem ser utilizados para configurar as interfaces de rede e os endereços IPs das máquinas virtuais. Os arquivos `.startup` são *scripts* que serão executados nas máquinas virtuais ao serem inicializadas. Na Figura B.5 segue um exemplo dos arquivos `.startup` que podem ser utilizados para configurar esse cenário de rede nas máquinas virtuais.

Estando no diretório do laboratório em um terminal de linha de comando, pode-se executar o comando `lstart` do Netkit e serão inicializadas todas as máquinas virtuais, com todas as suas conexões feitas e as configurações dos arquivos `.startup` realizadas, sendo possível pingar as máquinas virtuais que estão em uma mesma sub-rede. Para encerrar as máquinas virtuais pode ser utilizado o comando `lhalt`, que desliga corretamente as máquinas virtuais, ou o comando `lcrash`, que desliga as máquinas de forma instantânea. Ambos os comandos depois encerram os UML Switch utilizados.

```
#arquivo .startup da maquina virtual Host1
ifconfig eth0 10.0.0.1 netmask 255.255.255.0 up

#arquivo .startup da maquina virtual Host2
ifconfig eth0 10.0.0.2 netmask 255.255.255.0 up

#arquivo .startup da maquina virtual Host3
ifconfig eth0 15.0.0.1 netmask 255.255.255.0 up

#arquivo .startup da maquina virtual Host4
ifconfig eth0 15.0.0.2 netmask 255.255.255.0 up

#arquivo .startup da maquina virtual RotA
ifconfig eth0 10.0.0.254 netmask 255.255.255.0 up
ifconfig eth1 20.0.0.2 netmask 255.255.255.0 up

#arquivo .startup da maquina virtual RotB
ifconfig eth0 15.0.0.254 netmask 255.255.255.0 up
ifconfig eth1 20.0.0.1 netmask 255.255.255.0 up
```

Figura B.5: Exemplo dos arquivos `.startup` da rede utilizada no exemplo.

ANEXO C – Usando a ferramenta Graphviz e exemplo de uso

C.1 Usando Graphviz

Para se gerar imagens com a ferramenta Graphviz é preciso criar um arquivo texto que contêm a descrição do grafo que vai ser gerado a imagem, e após isso utilizar um dos programa que compõem o Graphviz em um terminal de linha de comando com as suas opções para a imagem ser gerada.

Um exemplo da descrição de um grafo simples, com apenas quatro vértices ligados entre si, formando um círculo, pode ser visto na Figura C.1, vamos considerar que o grafo esteja salvo em um arquivo texto chamado `grafo.gv`.

```
graph{
"1"--"2"--"3"--"4"--"1"
}
```

Figura C.1: Exemplo da descrição de um grafo simples.

A imagem do grafo pode ser gerado com qualquer um dos programas que compõem o Graphviz, o comando ‘‘`circo -Tpng grafo.gv -o imagem.png`’’ por exemplo, gerá a imagem utilizando o programa `circo`, salvando a imagem no formato PNG no arquivo `imagem.png`. As opções usadas podem podem sofrer várias alterações, o programa `circo` pode ser substituído por qualquer um dos outros quatro programas do Graphviz, a opção `-Tpng` pode ser modificado para salvar a imagem em vários formatos diferentes como GIF, JPEG, PDF, etc, a opção `-o` que define o arquivo que será salva a imagem pode ser substituído pela opção `-O`, onde não precisa ser definido um arquivo de saída para a imagem, será criado um arquivo com o mesmo nome do arquivo texto que contém a descrição do grafo acrescentado do formato da imagem gerada. A imagem do grafo gerada com o programo `circo` pode ser vista na Figura C.2.

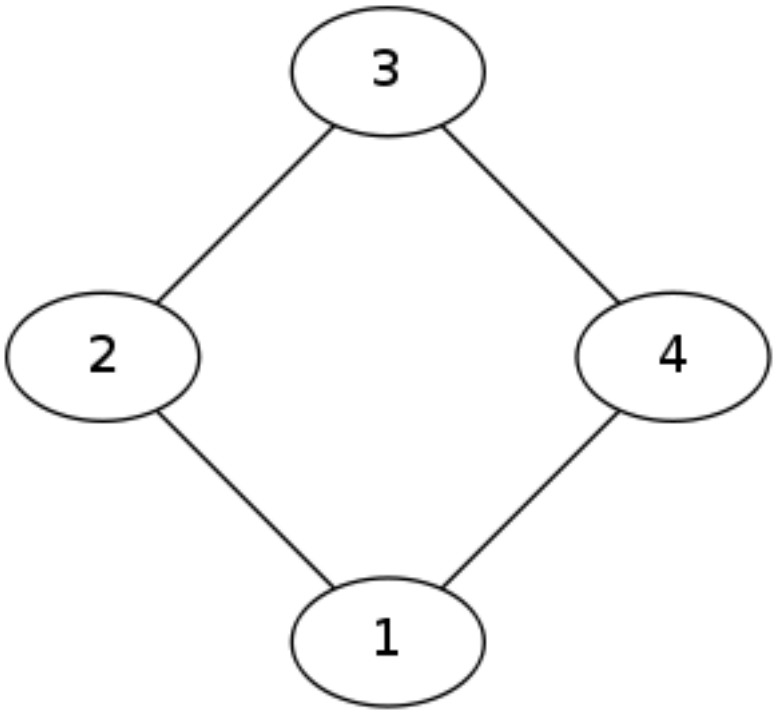


Figura C.2: Imagem gerada com o programa `circo` da ferramenta Graphviz.

A imagem gerada na Figura C.2 é um grafo normal, para ser gerado a imagem de um dígrafo, precisa ser mudado no arquivo texto a definição `graph` para `digraph`, e as ligações (arestas) entre os vértices ao invés de serem feitas utilizando a sintaxe `--`, são feitas utilizando a sintaxe `->`, indicando o sentido da aresta. Como já dito o Graphviz oferece uma grande variedade de opções para personalizar os grafos, mudando suas cores, formatos, permitindo escrever textos neles, como vai ser feita a orientação dos vértices e arestas na imagem, etc, algumas dessas opções funcionam para todos os programas que compõem o Graphviz, outras só trarão diferenças em usar-se com certos programas, ou com apenas um. As opções são usadas colocando-se entre colchetes, ao lado dos vértices e suas ligações descritos no arquivo texto que descreve o grafo. Essa grande variedade de opções e sintaxes podem ser olhadas no site do projeto Graphviz em `http://www.graphviz.org/`.

C.2 Exemplo de uso da ferramenta Graphviz

Será mostrado um exemplo de uso da ferramenta Graphviz, para gerar a imagem de um grafo mais complexo. Como exemplo será realizado um grafo para descrever um fluxograma, que é uma das utilidades da ferramenta Graphviz. No arquivo texto que descreve o grafo, foi preciso utilizar várias opções para personalizar os vértices, como mudar sua forma e cores. Essas opções são colocadas entre colchetes e do lado são colocados os vértices que serão con-

figurados com essas opções. É possível colocar opções entre colchetes do lado das linhas onde é feito as conexões entre os vértices, para personalizar as arestas. O arquivo texto que descreve o grafo para o fluxograma do exemplo, pode ser visto na Figura C.3.

```
digraph{

node[shape=box,style="filled,rounded",color=black,fillcolor=yellow]; Inicio;
node[label="Fim"]; fim;
node[shape=parallelogram,style=filled,color=black,fillcolor=darkseagreen1];
node[label="Receber nota p1"] p1;
node[label="Receber nota p2"] p2;
node[label="Receber nota p3"] p3;
node[label="Receber nota recuperecao"] notarec;
node[shape=box,style=filled,color=black,fillcolor=mediumpurple2];
node[label="Calcular media=(p1+p2+p3)/3"] media;
node[shape=diamond,style=filled,color=black,fillcolor=indianred3];
node[label="media > 6",height=1,width=2]; decmedia;
node[label="nota recuperacao > 6",height=1,width=3]; decrec;
node[shape=note,style=filled,color=black,fillcolor=indianred1,height=0.8,width
    =1.2];
node[label="Aprovado"]; apro;
node[label="Recuperacao"]; rec;
node[label="Reprovado"]; repro;
node[shape=circle,color=black,height=0.5,width=0.5,label="",style=""]; junc;

"Inicio"->"p1"
"p1"->"p2"
"p2"->"p3"
"p3"->"media"
"media"->"decmedia"
"decmedia"->"apro"[label="Sim"]
"decmedia"->"rec"[label="Nao"]
"rec"->"notarec"
"notarec"->"decrec"
"decrec"->"apro"[label="Sim"]
"decrec"->"repro"[label="Nao"]
"apro"->"junc"
"repro"->"junc"
"junc"->"fim"
}
```

Figura C.3: Arquivo que descreve o grafo para o fluxograma do exemplo.

A imagem gerada do fluxograma pode ser vista na Figura C.4, sendo que foi descrito no arquivo texto um dígrafo, já que o fluxograma precisa de indicação para saber sua ordem. Foi utilizado o programa `dot` do Graphviz para gerar a imagem, sendo o programa que obteve o melhor resultado para representar o fluxograma.

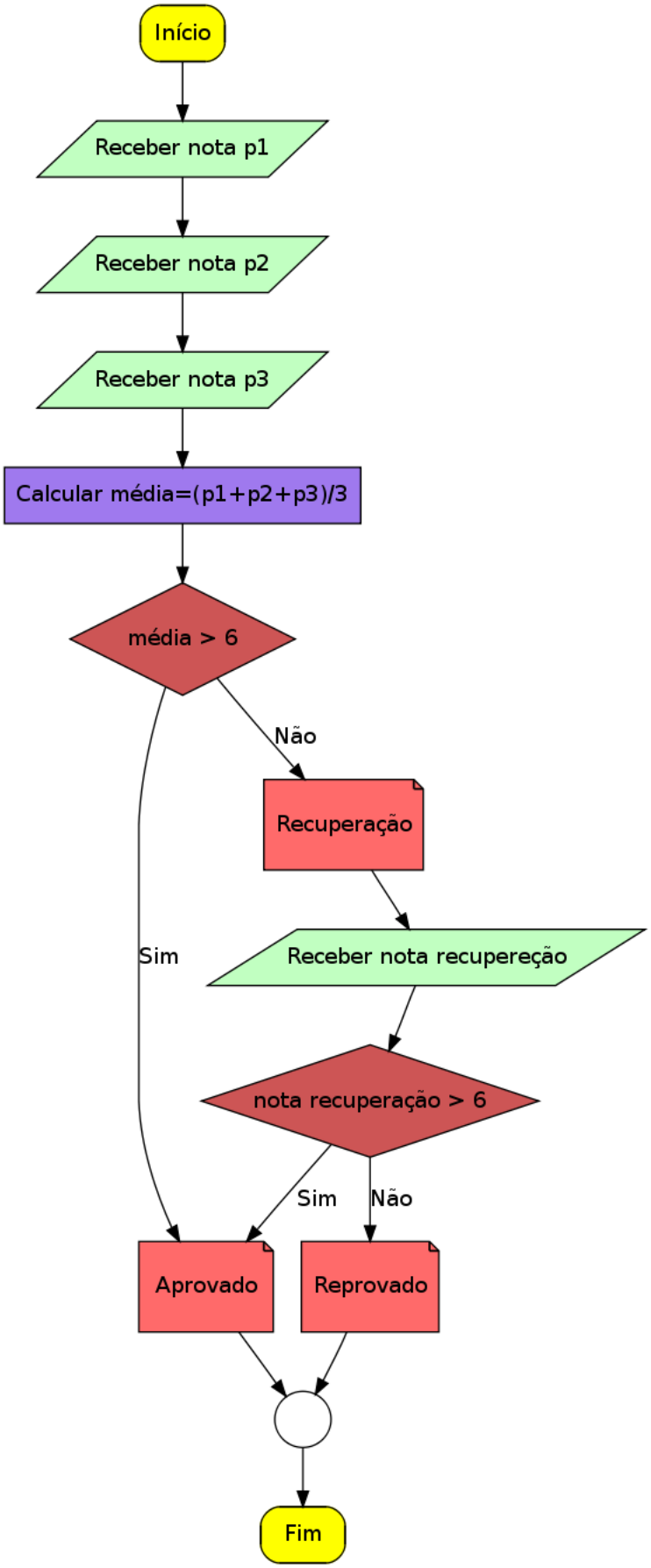


Figura C.4: Fluxograma gerado com o programa `dot` da ferramenta Graphviz.

ANEXO D – Preparar Laboratório do Netkit para ser usado com o Observador Didático

Primeiramente vamos considerar que já se tenha um laboratório do Netkit pronto, com o experimento que deseja-se observar suas mudanças nas tabelas de roteamento. Por exemplo, vamos considerar a rede mostrada na Figura D.1, juntamente com o seu arquivo `lab.conf`, já estando configurado um algoritmo de roteamento, por exemplo o RIP.

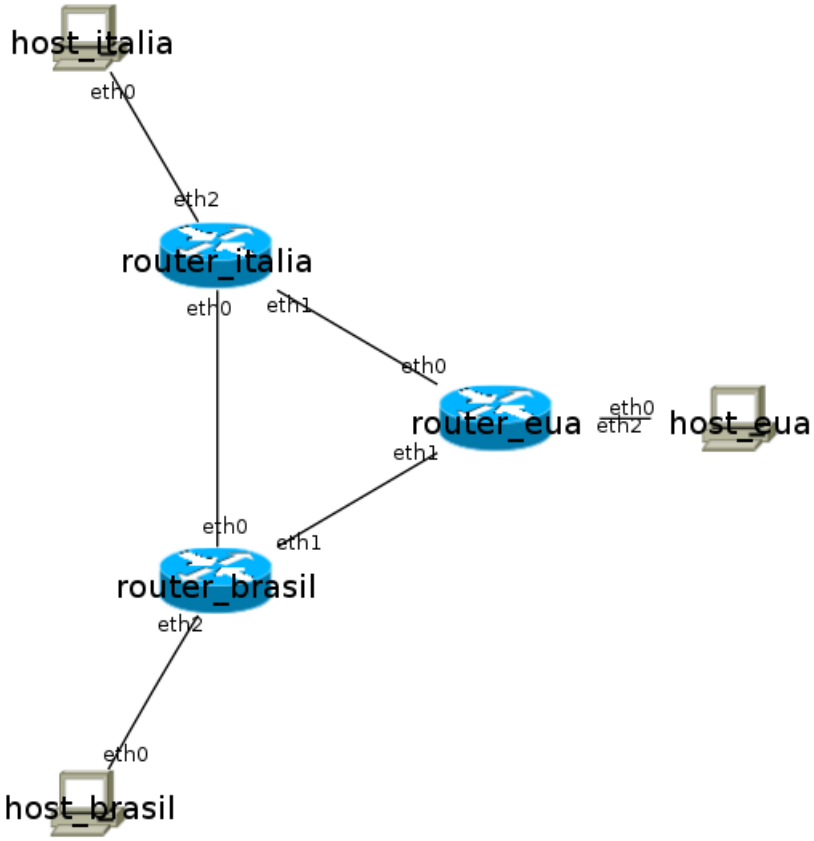


Figura D.1: Topologia de rede usada para o exemplo.

O arquivo `lab.conf` da rede da Figura D.1 pode ser visto na Figura D.2.

Primeiramente vamos começar alterando o arquivo `lab.conf`, já que o Observador Didático funciona capturando as tabelas de roteamento via protocolo SNMP e para isso precisa ter-se

```
1  host_brasil[0]="brasil"
2
3  host_eua[0]="eua"
4
5  host_italia[0]="italia"
6
7  router_brasil[0]="br-it"
8  router_brasil[1]="br-eua"
9  router_brasil[2]="brasil"
10
11 router_eua[0]="eua-it"
12 router_eua[1]="br-eua"
13 router_eua[2]="eua"
14
15 router_italia[0]="br-it"
16 router_italia[1]="eua-it"
17 router_italia[2]="italia"
```

Figura D.2: Arquivo `lab.conf` da rede da Figura D.1.

uma máquina na rede responsável por capturar as tabelas de roteamento, sendo essa maquina o gerente SNMP. Então vamos adicionar uma nova máquina no arquivo `lab.conf` e também mais uma interface de rede nos roteadores que se deseja ser monitorados ligando-os com o gerente SNMP, como mostrado na Figura D.3, destacando-se em amarelo as alterações feitas. Também já se aproveita e declara-se as máquinas virtuais que fazem parte do laboratório.

```
machines="router_brasil router_eua router_italia host_brasil host_eua host_italia gerente_snmp"

host_brasil[0]="brasil"

host_eua[0]="eua"

host_italia[0]="italia"

router_brasil[0]="br-it"
router_brasil[1]="br-eua"
router_brasil[2]="brasil"
router_brasil[3]="gerente"

router_eua[0]="eua-it"
router_eua[1]="br-eua"
router_eua[2]="eua"
router_eua[3]="gerente"

router_italia[0]="br-it"
router_italia[1]="eua-it"
router_italia[2]="italia"
router_italia[3]="gerente"

gerente_snmp[0]="gerente"
```

Figura D.3: Alterações no arquivo `lab.conf`.

Feito isso, ainda não foi adicionado uma nova máquina virtual ao laboratório, precisa ser criado ainda um diretório no laboratório com o mesmo nome da máquina adicionada e o seu arquivo `.startup`, responsável pelas configurações feitas no `boot` da máquina. Neste processo já se aproveita e também cria-se um diretório chamado `resultados` dentro do diretório do laboratório, onde o gerente guardará as mudanças das tabelas de roteamento em arquivos texto, um arquivo para cada roteador. A estrutura do laboratório após esse passo, pode ser vista na Figura D.4.

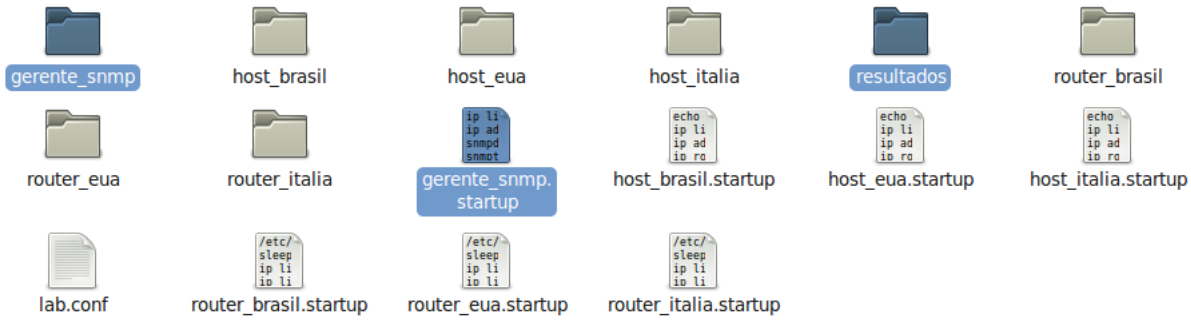


Figura D.4: Estrutura do laboratório após as modificações.

Foi criado o arquivo `.startup`, mas ele ainda está vazio, então vamos preencher ele com as configurações necessárias e os serviços que tem que ser inicializados no `boot` da máquina virtual que será o gerente SNMP. Primeiramente vamos colocar uma interface de rede e configurar seu IP, vamos lançar a interface de rede corresponde a mesma descrita no `lab.conf` que neste caso seria a eth0 e escolher uma sub-rede que ainda não foi usada na configuração dos outras interfaces usadas no laboratório. Assim o gerente e as interfaces extras adicionadas nos roteadores terão IPs diferentes. Também colocaremos os serviços que tem que ser inicializados no gerente que são o `snmpd` e o `snmptrapd`. Um exemplo de um arquivo `.startup` pode ser visto na Figura D.5, onde X.X.X.X seria um IP da sub-rede escolhida. Uma observação, existem outros jeitos de lançar uma interface e configurar seu IP, fica a critério de quem esta configurando o laboratório.

```
1 ip link set dev eth0 up
2 ip address add X.X.X.X/X dev eth0
3 snmpd
4 snmptrapd
```

Figura D.5: Exemplo de um arquivo `.startup`.

Já que estamos alterando os arquivos `.startup`, vamos alterar os arquivos .startup referentes aos roteadores que serão monitorados e que tiveram uma interface de rede extra adi-

cionada. Primeiramente vamos configurar e lançar a interface correspondente à adicionada no `lab.conf` e ter o cuidado que esteja na mesma sub-rede da interface do gerente SNMP. E como feito no gerente, vamos lançar os serviços e `scripts` que são necessários, sendo o serviço `snmpd` e o `script` responsável por ficar verificando se há alterações na tabela de roteamento e se houver alterações mandar uma `trap` SNMP para o gerente fazer a captura da tabela. O lançamento da interface de rede deve ser colocado juntamente onde já é feito as configurações das outras interfaces e logo abaixo o lançamento do serviço e o `script`, sendo X.X.X.X o endereço IP do gerente SNMP, para o roteador saber para onde mandar a *trap* SNMP. Preferencialmente configure-se as interfaces de rede e depois inicia-se os serviços e `scripts`, antes de lançar qualquer protocolo de roteamento, outras configurações seguem normalmente abaixo destes já feitas. Na Figura D.6 tem-se um exemplo de um arquivo `.startup` de uma máquina virtual que é um roteador.

```
1 #configurando outras interfaces de rede
2 ip link set dev eth3 up
3 ip address add Y.Y.Y.Y/Y dev eth3
4 #sendo Y.Y.Y.Y/Y um IP da mesma sub-rede do gerente snmp
5 snmpd
6 /etc/init.d/monitor.sh X.X.X.X &
7 #sendo X.X.X.X o endereco IP do gerente snmp
8 #continuando com outras configuracoes
```

Figura D.6: Exemplo de um arquivo `.startup` de uma máquina virtual que é um roteador.

Estamos iniciando esses serviços e `scripts`, mas os arquivos de configuração dos serviços e os próprios `scripts` ainda não existem, então vamos criá-los. Vamos começar fazendo isso para os roteadores. Dentro do diretório do laboratório cada maquina virtual tem um diretório com o seu mesmo nome, dentro desse diretório os arquivos e diretório colocados la dentro farão parte do sistema de arquivos da máquina virtual podendo ser usados pela máquina virtual. Dentro do diretório de cada roteador que será monitorado, vamos criar o diretório `etc`, e dentro desse diretório vamos criar os sub-diretórios `init.d` e `snmp` (se esses diretórios ainda não existirem). Dentro do diretório `init.d` iremos colocar o arquivo `monitor.sh` e dentro do diretório `snmp` iremos colocar o arquivo `snmpd.conf`, copiando esses arquivos dos arquivos padrões que já se tem, como pode ser visto nas Figuras D.7 e D.8.

O arquivo `monitor.sh` pode ser deixado do jeito que está, sem nenhuma modificação, funcionando corretamente para todos os roteadores. Já o arquivo `snmpd.conf` precisa ser feito pequenas modificações, que consistem praticamente de colocar o endereço IP do gerente em algumas linhas, o restante pode ficar do jeito que está. Na Figura D.9 segue as linhas que são necessárias modificar, onde X.X.X.X é o endereço IP do gerente SNMP.

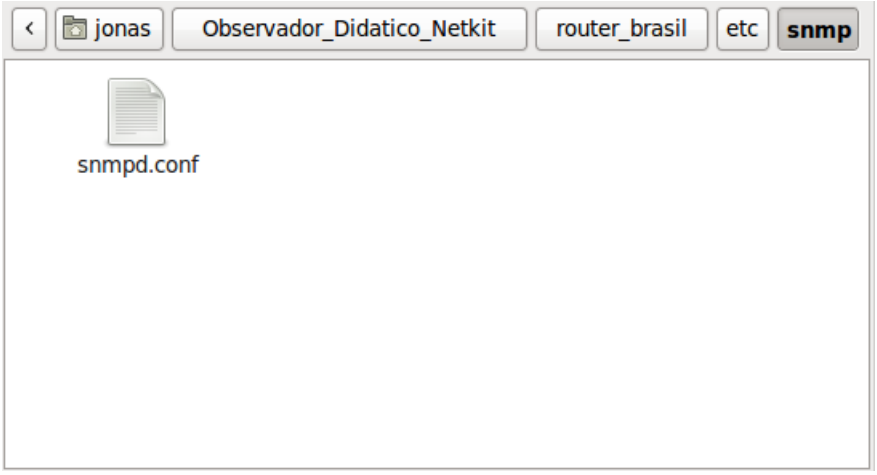


Figura D.7: Criando diretórios e arquivos necessários.

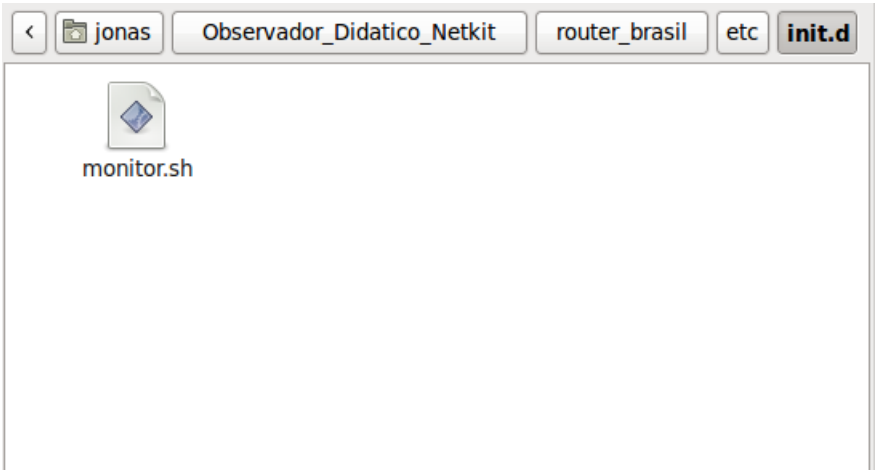


Figura D.8: Criando diretórios e arquivos necessários.

```
#outras linhas que nao sao necessario modificar
trap2sink X.X.X.X alunos 162
#outras linhas que nao sao necessario modificar
informsink X.X.X.X alunos 162
#outras linhas que nao sao necessario modificar
trapsess   X.X.X.X
#outras linhas que nao sao necessario modificar
```

Figura D.9: Modificações necessárias no arquivo `snmpd.conf`.

Depois de criado os arquivos para os roteadores, falta criar os arquivos para o gerente SNMP. Do mesmo modo feito para os roteadores, dentro do diretório do gerente SNMP no laboratório do Netkit, vamos criar o diretório `etc` e `tmp`. Dentro do diretório `etc` iremos criar o diretório `snmp`. Dentro do diretório `tmp` iremos colocar o arquivo `inFile.sh`, que é responsável pelo que fazer quando o gerente recebe uma `trap` vindo dos roteadores, que é capturar a tabela de roteamento via protocolo SNMP e salvar dentro da pasta `resultados` em um arquivo texto. Dentro da pasta `snmp` iremos colocar os arquivos `snmpd.conf` e `snmptrapd.conf`.

O arquivo `snmpd.conf` é o mesmo usado para os roteadores, sendo necessárias as mesmas modificações. Então pode-se copiar o mesmo arquivo já modificado usados nos roteadores. O arquivo `snmptrad.conf` não precisa nenhuma modificação, esse arquivo simplesmente tem a configuração que quando o gerente receber uma `trap` SNMP ele deve executar o `script` `inFile.sh`. Precisa ser feito modificações no arquivo `inFile.sh`, sendo o mais trabalhoso e mais chances de ocorrer erros. Esse arquivo consiste basicamente de ter vários IFs, um para cada roteador que será monitorado, sendo que os IFs são selecionados pelo endereço IP de quem mandou a `trap` SNMP, para o gerente saber de quem capturar a tabela de roteamento e salvar no arquivo texto correspondente. Um exemplo desse arquivo pode se visto na Figura D.10, sendo necessário ter um IF desses para cada roteador monitorado, e mudar as partes que estão destacadas em amarelo colocando o nome do roteador ou seu endereço IP.

É preciso lançar primeiro a máquina virtual do gerente SNMP para depois lançar as outras máquinas virtuais, assim o gerente já estaria pronto para receber as `traps` SNMP e capturar as tabelas. Isso pode ser feito através do arquivo `lab.dep` no diretório do laboratório do Netkit, criando-se esse arquivo e colocando a configuração que as máquinas virtuais só inicializam depois da máquina virtual do gerente SNMP ter sido inicializada. Essa configuração pode ser vista na Figura D.11.

Para o funcionamento correto do Observador Didático precisa ser criado no diretório `resultados`, os arquivos onde serão salvo as tabelas de roteamento, sendo um arquivo para cada roteador que é monitorado. Os aquivos terão o mesmo nome da máquina virtual seguidos de `.out`. Com isso o laboratório estrá preparado para ser usado junto com o Observador Didático, sendo possível ver as mudanças nas tabelas de roteamento dos roteadores.

inFile.sh

```
1 #!/bin/sh
2
3 read host
4 read ip
5 read val
6 read oid
7
8 ipe=`echo "$ip" | awk -F '[' '{ print $2 }' | awk -F ']' '{ print $1 }'`
9
10 #Filtrando mensagens do Roteador1:
11 if [ $ipe = IP Roteador1 ]
12 then
13         echo >> /hostlab/resultados/Nome Roteador1.out
14         hora=$(echo $oid | awk '{ print $2 }')
15         if [ `echo "$oid" | awk -F ':' '{ print $5 }'` != "coldStart" ]
16         then
17                 echo "Tabela de roteamento modificada às: $hora" >> /hostlab/resultados/Nome Roteador1.out
18                 snmpnetstat -On -v2c -c alunos -Crn IP Roteador1 >> /hostlab/resultados/Nome Roteador1.out
19                 echo "#" >> /hostlab/resultados/Nome Roteador1.out
20         fi
21 fi
22
23 #Filtrando mensagens do Roteador2:
24 if [ $ipe = IP Roteador2 ]
25 then
26         echo >> /hostlab/resultados/Nome Roteador2.out
27         hora=$(echo $oid | awk '{ print $2 }')
28         if [ `echo "$oid" | awk -F ':' '{ print $5 }'` != "coldStart" ]
29         then
30                 echo "Tabela de roteamento modificada às: $hora" >> /hostlab/resultados/Nome Roteador2.out
31                 snmpnetstat -On -v2c -c alunos -Crn IP Roteador2 >> /hostlab/resultados/Nome Roteador2.out
32                 echo "#" >> /hostlab/resultados/Nome Roteador2.out
33         fi
34 fi
```

Figura D.10: Exemplo do `script` `inFile.sh`.

```
1 router_brasil router_eua router_italia host_brasil host_eua host_italia:
      gerente_snmp
```

Figura D.11: Configuração do arquivo `lab.dep`.

Lista de Abreviaturas

UML *User-Mode Linux*

SNMP *Simple Network Management Protocol*

COW *Copy-On-Write*

IETF *Internet Engineering Task Force*

RFC *Request for Comments*

SMI *Structure of Management Information*

MIB *Management Information Base*

ISO *International Organization for Standardization*

ASN.1 *Abstract Syntax Notation 1*

EGP *Exterior Gateway Protocol*

RIP *Routing Information Protocol*

OSPF *Open Shortest Path First*

JVM *Java Virtual Machine*

Referências Bibliográficas

BATTISTA, G. D. et al. Netkit introduction. In: ____. [s.n.], 2007. Disponível em: <http://wiki.netkit.org/netkit-labs/netkit_introduction/netkit-introduction.pdf>. Acesso em: 10 de julho de 2011.

COMER, D. E. *Interligação de Redes com TCP/IP*. 5ª. ed. [S.l.]: Campus, 2006.

DIKE, J. *User Mode Linux*. [S.l.]: Prentice Hall, 2006.

FARIAS, R. *Observador didático do funcionamento de algoritmos e protocolos de roteamento*. 77 f. Monografia (Graduação) — Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina (IF-SC), Santa Catarina, 2010.

FARIAS, R.; CANTú, E. Observador didático do funcionamento de protocolos de reteamento. In: . [S.l.]: Seminário de Pesquisa do IF-SC, 2010.

GRAPHVIZ Theory. In: . [s.n.]. Disponível em: <http://www.graphviz.org/Theory.php>. Acesso em: 10 de julho de 2011.

KRASNYANSKY, M. Universal tun/tap device driver. In: ____. [s.n.], 2000. Disponível em: <http://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt>. Acesso em: 20 de Julho de 2011.

KUROSE, J.; ROSS, K. *Redes de computadores e a Internet: Uma abordagem top-down*. 3ª. ed. [S.l.]: Pearson Education, 2006.

MAURO, D.; SCHMIDT, K. *SNMP Essencial*. [S.l.]: O'Reilly, 2001.

STEVENS, W. R. *TCP/IP Illustrated, Volume 1: The Protocols*. 1ª. ed. [S.l.]: Addison Wesley, 2008.