UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE ESPECIALIZAÇÃO EM CONFIGURAÇÃO E GERENCIAMENTO DE SERVIDORES E EQUIPAMENTOS DE REDE

FELIPE WEISS MENINE

ESTUDO E IMPLEMENTAÇÃO DE INTERCONECTIVIDADE IPv6 ENTRE ROTEADORES LINUX E CISCO

MONOGRAFIA

CURITIBA
2014

FELIPE WEISS MENINE

ESTUDO E IMPLEMENTAÇÃO DE INTERCONECTIVIDADE IPv6 ENTRE ROTEADORES LINUX E CISCO

Monografia apresentada como requisito parcial para a obtenção do grau de Especialista em Configuração e Gerenciamento de servidores e equipamentos de rede, do Departamento Acadêmico de Eletrônica da Universidade Tecnológica Federal do Paraná – UTFPR

Orientador: Prof. MSc. Juliano de Mello Pedroso

CURITIBA

2014

RESUMO

MENINE, Felipe W. **Estudo e Implementação de Interconectividade IPv6 entre Roteadores Linux e Cisco.** 2014. 46 f. (V GESER - Especialização em Configuração e Gerenciamento de Servidores e Equipamentos de Redes). Universidade Tecnológica Federal do Paraná. Curitiba, 2014.

A presente Monografia descreve em detalhe o comportamento de uma rede de computadores interconectando dois roteadores de diferentes fabricantes - um Roteador Cisco com um PC Linux - em IPv6 utilizando o protocolo OSPFv3. De um lado um fabricante consagrado de equipamentos de rede e do outro, um roteador baseado em uma máquina de propósito geral com Software Livre. A ideia é provar a conectividade inter-redes, trazendo uma alternativa de baixo custo para uma rede com necessidade de expansão. Serão discutidos tópicos em redes de computadores, com ênfase no encaminhamento de pacotes em HW com Linux. O resultado é uma rede em perfeito funcionamento, descrita em um tutorial detalhado de instalação e configuração para ser usado como referência.

Palavras-chave: Redes IPv6. Protocolos de Roteamento. OSPFv3. Software livre. Linux.

ABSTRACT

MENINE, Felipe W. **Study and Implementation of IPv6 Interconnection between Linux and Cisco routers.** 2014. 64 pages. Monograph (Specialization in Configuration and Management of Servers and Network Equipments) - Federal Technological University of Paraná. Curitiba, 2014.

This Monograph describes in detail the behavior of a computer network interconnecting 2 routers from distinct vendors – a Cisco router and a Linux PC – in IPv6 using OSPFv3 protocol. On one hand, the well-known network equipment vendor and on the other, a router based on general-purpose PC using Open Source Operating System - Linux. The goal is to prove internetwork connectivity, bringing a low-cost alternative for an expanding network. Computer networks topics will be discussed with special attention to packet forwarding on Linux-installed HW. The outcome is a fully functional network, described in an in-depth tutorial for installation and configuration to serve as reference.

Keywords: IPv6 Networks. Routing Protocols. OSPFv3. Free Software. Linux.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Com o crescimento e popularização da internet, a comunicação de dados tornou-se praticamente indispensável nos ambientes corporativo e residencial, sendo que o Brasil chegou a marca de 73,9 milhões de internautas (INSTITUTO BRASILEIRO DE OPINIÃO PÚBLICA E ESTATÍSTICA, 2011). Transações bancárias, acesso a banco de dados de filiais a uma matriz e voz sobre IP (VoIP) são exemplos de facilidades providas do crescimento da internet.

A base para a comunicação em redes é o roteador, equipamento que interconecta redes distintas e torna possível a transmissão de informações entre equipamentos de uso final, tais como computadores, impressoras, telefones IP e até telefones (smartphones), entre outros.

Roteador é um dispositivo que encaminha pacotes de dados entre redes de computadores, criando um conjunto de redes de sobreposição. Um roteador é conectado a duas ou mais linhas de dados de redes diferentes. Quando um pacote de dados chega em uma das linhas, o roteador lê a informação de endereço no pacote. Em seguida, usando a informação de sua política aplicada em conjunto com sua tabela de roteamento ou encaminhamento, ele direciona o pacote para a rede de próxima em sua viagem. Os roteadores são os responsáveis pelo "tráfego" na Internet. Um pacote de dados é normalmente encaminhado de um roteador para outro através das redes que constituem a internet até atingir o nó destino.

Este trabalho reconhece e foca no roteador como um dispositivo de papel central em redes de computadores, que possibilita comunicação entre redes distintas.

1.1 OBJETIVOS

Nesta sessão serão trabalhados objetivo geral e objetivos específicos.

1.1.1 Objetivo Geral

O principal objetivo deste projeto é implantar uma rede IP de computadores através de roteador Cisco interconectado com roteador emulado em Linux e testar interconectividade entre as redes distintas. Ao final da atividade, elaborar um tutorial para configuração dos equipamentos para uso futuro.

1.1.2 Objetivos Específicos

- Montar fisicamente uma rede de computadores interconectados em topologia estrela com 1 roteador Cisco e um roteador Linux.
- Implementar, testar e validar funcionamento do protocolo de roteamento OSPFv3 entre equipamentos diferentes.
- Implementar, testar e comprovar habilidade da rede em suportar e rotear tráfego IPv6 puro.
- Elaborar um tutorial de configuração detalhado dos equipamentos, contendo imagens, scripts e printscreen das telas de configuração e resultados.

1.2 JUSTIFICATIVA

Ter uma rede IP local que permita uma variedade de hosts se comunicarem entre si e com a Internet implica necessariamente na instalação, configuração e administração de roteadores.

Hoje todas as pessoas e empresas precisam se conectar. Mas uma boa infraestrutura para atender a este requisito passa pela implementação de segurança e serviços avançados de rede, como ACLs, DHCP, NAT, Firewall, VPN entre outras. Isto consome recursos que muitas vezes são cruciais no início de uma atividade empresarial e que poderiam ser destinados ao cerne do negócio. Apesar da sua importância e produção em larga escala, o valor de um roteador comercial (como Cisco, HP, Juniper etc) nem sempre é justificável em redes menores que precisam de suas funcionalidades para garantir segurança e conectividade com a Internet.

A evolução dos sistemas baseados em software livre permitiu a concepção de um emulador de roteadores. O sistema operacional Linux por si só já atende a necessidades básicas de roteamento de pacotes IP, mas o surgimento de aplicações como Quagga e BIRD fomenta as necessidades de um cenário onde a conectividade de uma empresa é imperativa, mas os recursos financeiros são limitados inicialmente.

Supondo que, após algum tempo de operação, os recursos financeiros deixem de ser um impeditivo e a empresa possa então adquirir um roteador Cisco para sua rede. Ainda, supondo que mesmo após esta aquisição ainda haja necessidade de ter um segundo roteador (seja para fins de testes ou de isolamento de uma rede,por exemplo), e assim o roteador emulado deverá continuar na rede. Como se comportaria a conexão entre o roteador Cisco e o roteador Linux?

A falta de material acadêmico que demonstre as vantagens e desvantagens ou possíveis problemas ao se conectar roteadores Cisco com roteadores emulados é um dos grandes motivadores desta monografia.

Utilizando uma topologia de rede em estrela, pretende-se demonstrar que a rede opera normalmente. Pretende-se verificar o funcionamento de roteamento entre roteadores Cisco e Linux conectando diferentes redes.

Este trabalho visa também trabalhar com uma rede puramente IPv6, protocolo de comunicação que está substituindo o IPv4. Além da pressão da Anatel (Agência Nacional de Telecomunicações) e de outras entidades como o CGI.br (Comitê de Gestão da Internet) para modernização das atuais redes IP, a adoção do IPv6 traz algumas vantagens que são nativas do protocolo, discutidas mais a fundo na seção Desenvolvimento.

2 REFERENCIAIS TEÓRICOS

2.1 REDES DE COMPUTADORES

Uma Rede de computadores estabelece a interligação de computadores para o compartilhamento de recursos físicos ou lógicos. Esses recursos são normalmente informações (dados) em formato digital como encontradas em diretórios do disco rígido, DVD's, pendrive entre outros. A tecnologia de rede chegou ao estágio da massificação quando os computadores começaram a se espalhar pelo mundo comercial, ao mesmo tempo em que programas complexos multiusuários começaram a serem desenvolvidos (e-mail, banco de dados, Internet).

Na década de 1970 a ISO (International Organization for Standardization) criou um padrão universal para troca de informações intra e inter redes de computadores: o Modelo de Referência OSI, estabelecido em sete camadas, o qual incentivou a padronização de redes de comunicação e controle de processos distribuídos.

Um fato importante a ser considerado quanto ao padrão OSI foi o seu longo tempo para a sua definição. Durante esse período, o Departamento de Defesa do Governo dos Estados Unidos da América (DoD – Department of Defense) desenvolveu o Modelo de Referência TCP/IP, estabelecido em quatro camadas com o objetivo principal de manter conectados seus equipamentos.

Como alguns fabricantes iniciaram o desenvolvimento de equipamentos seguindo esse padrão, quando o padrão OSI foi finalizado muitos equipamentos já estavam funcionando no Modelo de Referência denominado TCP/IP. Assim, o Modelo de Referência OSI que nasceu para padronização foi preterido pelo que mercado já havia adotado: o modelo TCP/IP. As instituições acadêmicas não aceitaram substituir seus equipamentos, pois isto demandaria um alto custo e muito tempo perdido para treinamento e novas configurações.

O nome TCP/IP refere-se a uma pilha de protocolos que tem como principais protocolos o TCP (Transmission Control Protocol) e o IP (Internet Protocol) além de outros protocolos conhecidos tais como ARP, RARP, UDP e ICMP. Logo não devemos confundir a pilha de protocolos TCP/IP com os protocolos TCP e o protocolo IP, que possuem características de funcionamento bem distintos um do

outro. Hoje o modelo TCP/IP é a base para a Internet e também redes de computadores locais (privadas).

2.2 O CONJUNTO DE PROTOCOLOS TCP/IP

A arquitetura do TCP/IP pode ser vista na Figura 1

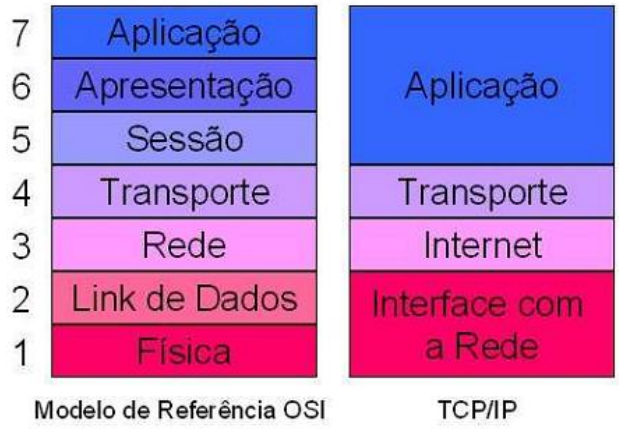

Figura 1 - comparativo entre camadas modelo OSI e TCP/IP

O TCP/IP tem quatro camadas. Na camada de Aplicação encontraremos os protocolos que os softwares de aplicação utilizam, tais como o SMTP (para e-mail), o FTP (para a transferência de arquivos) e o HTTP (para navegação web). Cada tipo de programa se comunica com um protocolo de aplicação diferente, dependendo da finalidade do programa (HUNT, 2002).

Após processar a requisição do programa, o protocolo na camada de Aplicação se comunicará com o protocolo na camada de Transporte, normalmente o TCP. Esta camada é responsável por pegar os dados enviados pela camada superior, dividi-los em segmentos e enviá-los à camada imediatamente inferior, a camada Internet. Este processo é chamado Encapsulamento. Além disso, durante a recepção dos dados, esta camada é responsável por colocar em ordem os pacotes recebidos da rede (já que podem chegar fora de ordem) e também verificar sua integridade.

Na camada Internet o IP (Internet Protocol, Protocolo Internet), que pega os pacotes recebidos da camada de Transporte e adiciona informações de endereçamento virtual, isto é, adiciona o endereço do computador que está

enviando os dados e o endereço do computador que receberá os dados. Esses endereços virtuais são chamados endereços IP. Em seguida os pacotes são enviados para a camada imediatamente inferior, a camada Interface com a Rede. Nesta camada os pacotes são chamados *frames* ou quadros.

A camada Interface com a Rede receberá os pacotes enviados pela camada Internet e os transmitirá para a rede (ou receberá os dados da rede, caso o computador esteja recebendo dados). Atualmente, nesta camada praticamente todos os computadores utilizam o padrão Ethernet, que compreende 2 sub-camadas: Controle do Link Logico (LLC) e Controle de Acesso ao Meio (MAC), listadas de cima para baixo. A Figura 2 demonstra como os dados são encapsulados para então serem transmitidos para a rede.

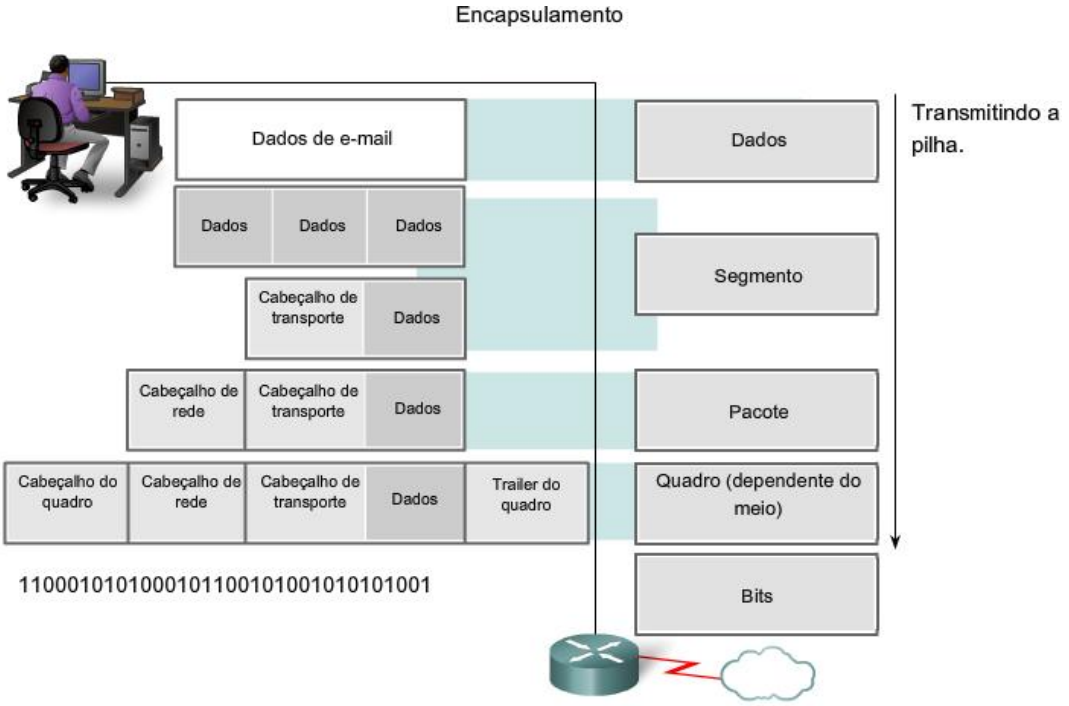

Figura 2 - Encapsulamento de dados na rede Fonte: (Cisco Systems, Inc., 2009) módulo 1, cap. 3

A Tabela 1 resume as características das camadas do modelo TCP/IP:

CAMADA MODELO TCP/IP	PROTOCOLO	NOMENCLATURA DA INFORMAÇÃO	CARACTERÍSTICAS	EXEMPLOS
Aplicação	HTTP, SMTP, FTP, TELNET	Dados	Mais próximo do usuário. Dependem do software utilizado.	Navegação WEB, Email, Transferência de arquivos, conexão remota
Transporte	TCP	Segmento	Orientado a conexão; controle de fluxo	Ping (ICMP), HTTP, HTTPS
Transporte	UDP	Segmento	Não orientado a conexão	(DNS); Vídeo em Streaming; Voz Sobre IP (VOIP)
Internet	IP	Pacote ou Datagrama	Endereço de Origem e destino	
Conexão ao meio	Ethernet	Frame e trailer	Frame: end. Físico de origem e destino Trailer: informações para verificação de erros	

Tabela 1 - Características das camadas do modelo TCP/IP

As camadas que compõem os modelos OSI e TCP/IP se dividem em: camadas de Fluxo de Dados e Camadas de Aplicativos. A Figura 3 traz um comparativo entre os 2 modelos:

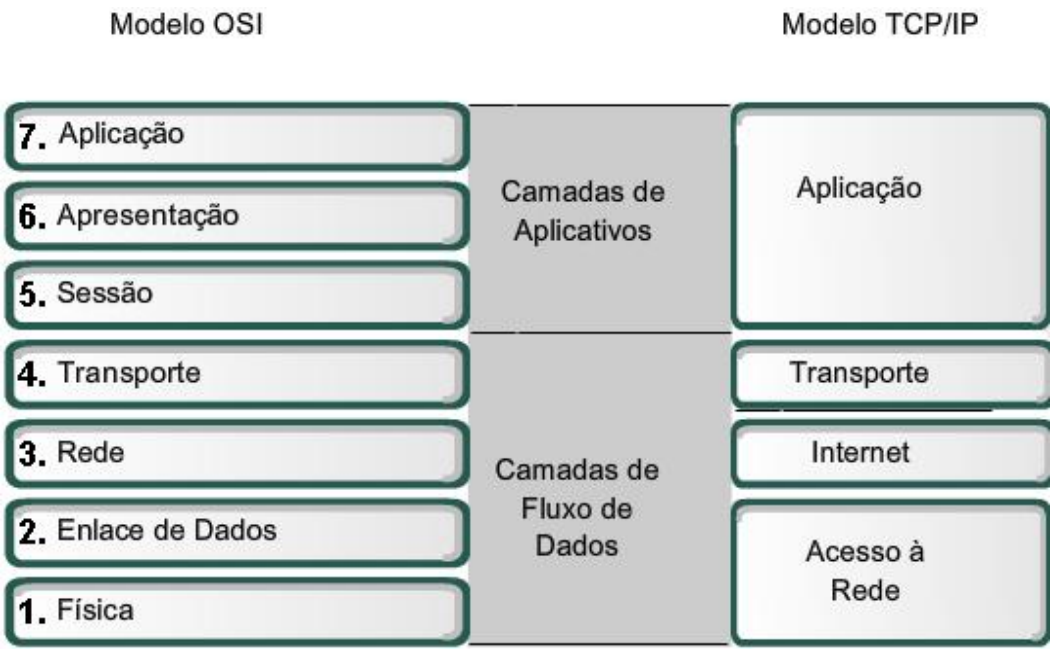

Figura 3 - tipos de camadas

Camadas de aplicativos se encarregam de mostrar a informação “palpável” para o usuário da rede e possibilitar serviços como email e navegação web.

Camadas de Fluxo de Dados são responsáveis pelo trânsito da informação através de diferentes redes e protocolos. É nestas camadas que reside o foco deste trabalho, em especial na camada Internet do modelo TCP/IP.

Nas seções a seguir, serão introduzidas as camadas e os protocolos TCP/IP:

2.3 CAMADA DE APLICAÇÃO

Esta camada faz a comunicação entre os programas e os protocolos de transporte. Existem vários protocolos que operam na camada de aplicação. Os mais conhecidos são o HTTP (*HyperText Transfer Protocol*, Protocolo de Transferência Hipertexto), o SMTP (*Simple Mail Transfer Protocol*, Protocolo Simples de Transferência de Correspondência), o FTP (*File Transfer Protocol*, Protoloco de Transferência de Arquivos), o SNMP (*Simple Network Management Protocol,* Protocolo Simples de Gerenciamento de Redes), o DNS (*Domain Name System*, Sistema de Nome de Domínio) e o Telnet.

A camada de aplicação comunica-se com a camada de transporte através de uma porta. As portas são numeradas e as aplicações padrão usam sempre uma mesma porta. Por exemplo, o protocolo SMTP utiliza sempre a porta 25, o protocolo HTTP utiliza sempre a porta 80 e o FTP as portas 20 (para transmissão de dados) e 21 (para transmissão de informações de controle).

2.4 CAMADA DE TRANSPORTE

Na transmissão de dados, a camada de transporte é responsável por pegar os dados passados pela camada de aplicação e encapsulá-los em pacotes. O TCP (*Transmission Control Protocol*, Protocolo de Controle da Transmissão) é o protocolo mais usado na camada de Transporte.

Na recepção de dados, o protocolo TCP recebe e ordena os pacotes da camada Internet confere se os dados dentro dos pacotes estão íntegros e envia um sinal de confirmação chamado ”*acknowledge*” (ACK) ao transmissor, avisando que o pacote foi recebido corretamente e que os dados estão íntegros. Se nenhum sinal de confirmação (ACK) for recebido (ou porque o dado não chegou ao destino ou porque o TCP descobriu que dado estava corrompido), o transmissor enviará novamente o pacote perdido.

Enquanto que o TCP reordena os pacotes e usa mecanismos de confirmação de recebimento - o que é desejável na transmissão de dados - existe outro protocolo que opera nesta camada que não tem esses recursos: o protocolo UDP (*User Datagram Protocol* - Procolo de Datagrama do Usuário).

Por essa razão o TCP é considerado um protocolo confiável, enquanto que o UDP é considerado um protocolo não confiável. O UDP é tipicamente usado quando nenhum dado importante está sendo transmitido, como requisições DNS. Como o UDP não reordena os pacotes nem usa mecanismo de confirmação, ele é mais rápido que o TCP.

2.5 CAMADA INTERNET

Nos protocolos TCP/IP, cada computador é identificado com um endereço virtual único, chamado endereço IP. A camada Internet é responsável por adicionar um cabeçalho ao pacote de dados recebidos da camada de Transporte onde, entre outros dados de controle, será adicionado também o endereço IP de origem e o endereço IP de destino, isto é, o endereço IP do computador que está enviando os dados e o endereço IP do computador que deverá recebê-los.

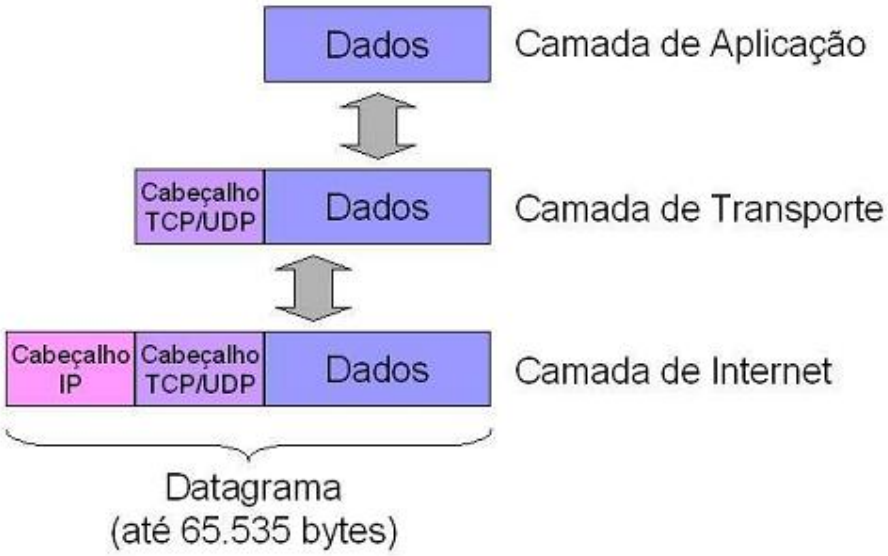

Figura 4 - Cabeçalhos e camadas

Os pacotes de dados são enviados usando o protocolo IP.

O IP pega os segmentos de dados recebidos da camada de Transporte (do protocolo TCP se você está transmitindo dados como e-mails ou arquivos) e os divide em pacotes, que são transmitidos sem aguardar confirmações do tipo ACK.

2.6 CAMADA INTERFACE COM A REDE

Os datagramas gerados na camada Internet serão passados para a camada Interface com a Rede, durante a transmissão de dados, ou a camada de Interface com a Rede pegará os dados da rede e os enviará para a camada de Internet, na recepção dos dados. Esta camada é definida pelo tipo de rede física a qual o

computador está conectado. Quase sempre o computador estará conectado a uma rede Ethernet.

O TCP/IP é um conjunto de protocolos que lida com as camadas 3 a 7 do modelo de referência OSI, enquanto que o Ethernet é um conjunto de protocolos que lida com as camadas 1 e 2 do modelo de referencia OSI - o que significa que o Ethernet lida com os aspectos físicos da transmissão de dados. Por isso um complementa o outro, já que precisamos das sete camadas completas (ou suas equivalentes) para estabelecer uma conexão de rede.

2.7 ROTEAMENTO IP

O que o IP faz?

A informação de cabeçalho de um pacote IP contém toda informação necessária para possibilitar algumas funções críticas de rede. Estas funções incluem:

- Endereçamento e roteamento
- Fragmentação e remontagem
- Detecção e correção de dados danificados pelo trânsito

A camada IP conta com o hardware de rede subjacente para sua transmissão. Isso significa que o pacote IP é encapsulado pelos quadros da camada inferior, como Ethernet, Token Ring ou X25.

Uma das funções mais cruciais de uma rede IP é o roteamento. Roteamento é o processo de descobrir, comparar e selecionar caminhos através da rede para qualquer endereço IP de destino. Tipicamente, a função de roteamento é incorporada por equipamentos específicos chamados roteadores. Todavia o avanço tecnológico está rapidamente obscurecendo as distinções entre roteadores tradicionais, switches LAN e até hosts. Hoje, todos os 3 equipamentos são capazes de descobrir, comparar e selecionar rotas. Portanto, roteamento deve ser entendido como uma função, não como equipamento físico. (SIYAN & PARKER, 2002)

A essência do roteamento existe na forma de protocolos de rede altamente especializados que habilitam roteadores (independentemente de que tipo de

equipamento físico este possa ser) para executar suas funções vitais. Estas funções incluem

- Compartilhar informações sobre redes e hosts conectados localmente
- Comparar caminhos potencialmente redundantes
- Convergência após acordo da topologia de rede.

Entendendo os mecanismos dessas funções básicas em uma rede IP, poderemos examinar com mais profundidade três dos protocolos de roteamento IP dinâmicos mais comumente encontrados.

Os fundamentos do Roteamento

Roteadores podem rotear basicamente de duas formas. Podem usar rotas estáticas pré-programadas ou podem calcular rotas usando um dos vários protocolos de roteamento dinâmico. Estes são usados pelos roteadores para descobrir rotas. Os roteadores então encaminham pacotes para estas rotas. Um roteador programado apenas com rotas estáticas é incapaz de descobrir rotas: falta-lhes um mecanismo para comunicar informações de roteamento com outros roteadores. Desta forma os pacotes somente serão encaminhados para rotas definidas pelo administrador de rede.

Além da programação estática de rotas, há duas categorias básicas de protocolos de roteamento dinâmico:

- Vetor-distância
- Estado de link

A principal diferença entre estes dois tipos de protocolo está na maneira com que descobrem e calculam novas rotas para os destinos.

Roteamento estático

Rotas estáticas ou pré-programadas são a forma mais simples de roteamento. As tarefas de descobrir rotas e propaga-las através da rede são

deixadas aos administradores de rede. Um roteador programado para roteamento estático encaminha pacotes para portas predeterminadas.

Após a relação entre o endereço IP de destino e a porta do roteador a ser usada, não há mais necessidade para roteadores tentarem descobrir rotas ou mesmo comunicarem informações sobre rotas para aquele destino. É possível, entretanto que o roteador use rotas estáticas para alguns destinos e dinâmicas para outros.

Há muitos benefícios para uso de rotas estáticas. Por exemplo segurança de redes: sem a configuração de protocolos de roteamento dinâmico, nenhum outro roteador “malicioso” conectado à rede será capaz de interferir numa tabela de roteamento estática, induzindo a caminhos inseguros pela rede. Outra questão é que roteamento estático utiliza muito menos largura de banda das transmissões disponíveis, não desperdiça capacidade de processamento do roteador tentando calcular rotas, e necessita de menos memória. Assim, para algumas redes é possível utilizar roteadores menores e mais baratos. Apesar destes benefícios, há limitações inerentes ao roteamento estático que precisamos considerar.

Rotas estáticas são usadas normalmente para redes pequenas cuja topologia dificilmente muda. Um exemplo são dois sites conectados por um link ponto a ponto. Outro uso pode ser para solução de problemas e correções temporárias para problemas de roteamento (no caso de uma tabela de roteamento corrompida).

Normalmente a mistura de roteamento estático e dinâmico não é interessante. Isto pode levar a rotas estáticas conflitarem com rotas computadas dinamicamente. Neste caso as rotas estáticas são utilizadas mesmo que a rota dinâmica seja mais eficiente.

2.8 ROTEAMENTO COM VETOR DISTÂNCIA

No roteamento baseado em algoritmos de vetor distância, o algoritmo periodicamente passa cópias de suas tabelas de roteamento aos seus vizinhos imediatos na rede. Cada recipiente adiciona um vetor de distância, ou seu próprio “valor” de distância à tabela de roteamento e então a encaminha aos seus vizinhos imediatos. Este processo ocorre de forma omnidirecional entre roteadores

imediatamente vizinhos. Este processo passo-a-passo resulta em cada roteador aprendendo sobre outros roteadores e desenvolvendo uma perspectiva cumulativa de “distâncias” de rede.

A tabela cumulativa então é usada para atualizar a tabela de roteamento de cada roteador. Quando completado o processo, cada roteador aprendeu valores de “distâncias” a recursos da rede, mas não aprendeu nada específico sobre outros roteadores ou sobre a real topologia da rede.

2.8.1 Desvantagens do roteamento com Vetor-distância:

Este tipo de roteamento pode criar problemas sob algumas circunstâncias. Por exemplo, uma falha ou mudança na rede requer um tempo para convergência dos roteadores em um novo entendimento da topologia. Durante este tempo, a rede pode estar vulnerável a roteamento inconsistente, loops etc.

2.8.2 Benefícios do roteamento com Vetor-distância:

Protocolos de vetor-distância são geralmente muito simples, fáceis de entender, configurar, manter e usar. Consequentemente são muito úteis em pequenas redes com poucos caminhos redundantes e de baixa exigência em performance.

O RIP (Routing Information Protocol) é o exemplo típico de roteamento com vetor-distância, sendo amplamente utilizado há décadas. Ele usa uma única métrica para determinar o melhor caminho para envio de qualquer pacote: custo.

2.9 ROTEAMENTO LINK-STATE

Algoritmos de Roteamento link-state mantém uma complexa base de dados da topologia de rede. Diferente dos protocolos vetor-distância, protocolos link-state desenvolvem e mantém um conhecimento completo dos roteadores e de como se interconectam. Isso ocorre através da troca de Anúncios Link-State com outros

roteadores da rede, com os quais se constrói uma base topológica de dados. Um algoritmo de Link-state então é usado para computar a acessibilidade a destinos de rede. Esta informação então é usada para atualizar a tabela de roteamento. Este processo é capaz de descobrir alterações na topologia de rede causadas pela falha de componentes ou crescimento da rede.

Na realidade, a troca de anúncios Link-state é ocorre por um evento na rede ao invés de rodar periodicamente como no RIP. Isso acelera muito o processo de convergência pois não é necessário aguardar uma série de temporizadores arbitrários para que os roteadores possam convergir.

2.9.1 Desvantagens do roteamento link-state

Durante o processo inicial de descobrimento, os protocolos de roteamento link-state podem inundar os meios de transmissão da rede, diminuindo assim a capacidade da rede em transportar dados temporariamente (mas sensivelmente).

Roteamento link-state é intense em memória e processamento, exigindo roteadores mais equipados que consequentemente são mais caros.

2.9.2 Vantagens do roteamento link-state

A abordagem link-state de roteamento dinâmico pode ser muito útil em redes de qualquer tamanho. O roteamento link-state capacita uma rede bem desenhada a superar os efeitos de alterações topológicas inesperadas. O uso de eventos para disparar atualizações (ao invés de temporizadores fixos) permite uma convergência muito mais rápida e libera banda para tráfego de dados ao invés de tráfego de manutenção.

Outro efeito da eficiência da largura de banda em roteamento link-state é facilitar a escalabilidade da rede mais que em roteamento estático ou de vetor distância. Levando em conta as limitações destes 2 ultimos, é natural concluir que o roteamento link-state é melhor para redes maiores e mais complexas ou que precisam ser muito escaláveis.

A complexidade da configuração inicial do roteamento link-state em uma grande rede pode ser mais desafiadora, mas compensa no longo prazo.

2.10 ROTEAMENTO POR SOFTWARE BASEADO EM PC COM SOFTWARE LIVRE

A maioria dos roteadores usa hardware customizado para rotear dados entre redes de computadores. Roteadores por software executam as mesmas tarefas usando hardware commodity (computadores pessoais comuns) imitando o comportamento de um hardware de roteador em um software. (Mims, 2010)

A escolha em construir roteadores IP com hardware padrão “off-the-shelf” deriva da ampla disponibilidade de documentação, baixo custo associado à produção em larga escala e a contínua evolução induzida pelo mercado. Por outro lado, software de código aberto proporciona a oportunidade de modificar a operação do roteador para adaptação a qualquer necessidade. (BIANCO, FINOCHIETTO, GALANTE, MELLIA, & NERI, 2005).

Arquiteturas de Roteadores por Software (*Software Routers* ou SR) baseadas em software de código aberto fornecem altos níveis de flexibilidade e modularidade, sendo consideradas promissoras para desenvolvimento de nós de rede multicamadas. Além disso, quando implementadas em HW de PC, tais soluções de SR geralmente possuem atributos adicionais atrativos como disponibilidade multi-vendor, baixo custo e habilidade para atualizar peças básicas para facilitar a customização. (BOLLA & BRUSCHI, 2008).

Nos ultimos anos, o interesse crescente neste tipo de equipamento guiou alguns projetos conhecidos como Quagga, Xorp, BIRD entre outros, projetados para desenvolver plataformas completas de roteamento IP sobre Sistemas Operacionais (SO) modernos de código aberto com recursos de rede avançados (por ex. Linux e OpenBSD).

Apesar do potencial e flexibilidade em desenvolver novas funcionalidades e mecanismos, assim como a disponibilidade de SW de rede consolidado, críticas são comuns aos SR principalmente quanto à performance no plano de encaminhamento (ou plano de dados em algumas fontes). Estas ressalvas normalmente surgem da comparação entre SRs, que são baseados em HW de uso geral, e equipamentos

comerciais, onde as operações mais intensas e críticas quanto ao tempo são normalmente atribuídas a componentes de HW customizados, como processadores de rede ou chips ASIC/FPGA. Hoje, o gap de performance pode não ser tão grande e de qualquer forma é mais do que justificado em muitos cenários tanto pela diferença de custo quanto pela flexibilidade oferecida por uma solução de SW.

Este trabalho irá focar em software de roteamento baseado em Linux, portanto este é o sistema operacional abordado.

2.10.1 O Plano de Encaminhamento no Linux

A arquitetura de processamento de pacotes do Linux (e outros SOs) é geralmente iniciada por interrupções de HW: a placa de rede sinaliza o kernel na recepção ou transmissão de pacote através de interrupções de HW. Cada interrupção de HW (*interrupt request* ou IRQ) é servida o mais breve possível por uma rotina de manuseio, que suspende as operações sendo processadas pela CPU no momento. Os *tratadores de interrupção* (ou *IRQ handlers* que são rotinas de serviço de interrupção) são projetados para serem muito curtos, enquanto todas as tarefas demoradas são executadas mais tarde pelas chamadas “Software IRQs” (SoftIRQs). SoftIRQs se diferem de HW IRQs principalmente por serem programadas (agendadas) para execução por uma atividade de kernel (como por ex. uma rotina de HW IRQ), e tem que esperar até que seja chamada pela atividade “programadora” (que a agendou). SoftIRQs somente podem ser interrompidas por rotinas de HW IRQ.

O processo de encaminhamento é disparado por uma HW IRQ gerada por um dispositivo de rede que sinaliza a recepção ou transmissão de pacotes. Depois, a rotina correspondente executa algumas checagens rápidas e marca a correta SoftIRQ, que é ativada pelo agendador de tarefas (escalonamento de processos) do kernel assim que possível. Quando a SoftIRQ é finalmente executada, ela executa todas as operações de processamento de pacotes acessando duas regiões especiais da memória RAM, chamadas TxRing e RxRing, que são usadas pelo DMA para transferir ou receber pacotes da placa de rede. DMA é *Direct Memory Access* ou Acesso Direto à Memória – recurso que permite que alguns dispositivos de HW acessem a memória principal do sistema independentemente da CPU.

Em particular, uma SoftIRQ executa duas tarefas principais na placa de rede, ligadas a recepção e transmissão de pacotes, respectivamente. A primeira tarefa é a desalocação de pacotes já transmitidos do TxRing, enquanto a segunda tarefa inclui todas as operações de encaminhamento de pacotes na prática. Na verdade, a segunda tarefa cuida dos pacotes recebidos no RxRing com o procedimento NAPI e executa todas as operações seguintes de entrega em camadas 2 e 3 (L2 e L3) até a decisão de roteamento e até o enfileiramento no TxRing da interface de saída selecionada. NAPI ou (*New API*) é um procedimento para atrasar o manuseio de pacotes de entrada até que se tenha quantidade suficiente que valha a pena o tratamento de todos os pacotes uma só vez. Tal abordagem tem objetivo de mitigar interrupções e reduzir o *overhead* de recebimento de pacotes.

Com relação ao Linux e suporte Multi-CPU nas operações do plano de encaminhamento, os kernels 2.6 e superiores (utilizaremos o Debian 7.1 “Wheezy” com Linux Kernel v3.2) alcançaram um alto nível de optimização. A chave para esta melhoria lida principalmente com otimização e refinamento do travamento de código (para serializar acessos em dados compartilhados), e a capacidade de amarrar um processo/atividade a uma CPU específica, chamado “afinidade de CPU”, assim mantendo uso eficaz do cache do processador.

Além disso, o kernel do Linux permite designar certos IRQs a específicos CPUs/cores através da chamada “afinidade de IRQ”. Isto tem forte impacto no plano de encaminhamento do *Software Router* posto que permite controle de qual CPU/core deve processar as rotinas de tratamento de HW IRQ (e as seguintes SoftIRQs) para a placa de rede correspondente. Assim, cada CPU/core executa tanto a desalocação de pacotes transmitidos pela placa de rede amarrada quanto as operações de encaminhamento para todos os pacotes recebidos pela mesma placa.

2.10.2 Gargalos Arquiteturais

Quando é usada a arquitetura de Software Router baseada em uma única CPU/core, a performance do plano de encaminhamento pode ser substancialmente limitada por apenas 2 fatores chave: a capacidade computacional do SR e a largura de banda / latência dos Bus de I/O. Enquando esta última afeta a o *throughput*

máximo em termos de bits/s, a primeira limita a taxa de cabeçalhos de pacote que podem ser processados pela CPU.

Com respeito a isso, enquanto a adoção da PCI-express proporciona grande largura de banda I/O, arquiteturas multiprocessadores como SMP (*Symmetric Multi-Processing* ou multiprocessamento simétrico como é o caso Intel Core) incrementam consideravelmente a capacidade computacional no geral, pois permite processamento paralelo em múltilpas CPU/core. Entretanto, para explorar efetivamente este tipo de plataforma de HW, é necessária uma arquitetura de SW capaz de paralelizar a carga computacional o máximo possível.

Problemas de performance típicos que possam minar o ganho de paralelismo surgem quando tarefas em CPUs diferentes compartilham alguns dados. Isto geralmente leva a dois problemas: serialização de acesso aos dados e coerência de cache de CPU. Ambos introduzem tempos de latência custosos no acesso e processamento de dados compartilhados.

Começando da descrição do plano de encaminhamento na seção anterior, pode-se ver que TxRings claramente representam estruturas de dados compartilhadas entre SoftIRQs rodando em todas as CPUs/cores envolvidas no encaminhamento de pacotes. Na realidade, o TxRing de uma placa específica é preenchido com pacotes vindos de por todas as CPUs/cores entregando tráfego em direção àquela placa, enquanto é servida somente pela CPU específica ligada ao seu HW IRQ.

Em detalhe, os acessos do SoftIRQ a cada TxRing são serializados por um procedimento de trava de código (mecanismo de sincronização de acesso a recursos), que é baseado na função “*spinlock*”, na qual a instrução fica aguardando (e constantemente checando) disponibilidade para execução. Esta trava (chamada trava “LLTX”) garante que cada TxRing possa ser lido ou modificado por apenas um SoftIRQ por vez. Quando um SoftIRQ ganha acesso ao TxRing, ele pára temporariamente os SoftIRQs seguintes, os quais devem aguardar a disponibilidade do TxRing. A contenção da trava LLTX obviamente causa desperdício computacional (devido às vezes de espera da trava), especialmente quando muitos CPUs/cores são envolvidos no processo de encaminhamento.

Além disso, outro desperdício de performance é causado por gerenciamento de cache de CPU/core: dados compartilhados podem ser mantidos apenas no cache de um processador por vez, caso contrário o cache da CPU/core perder sincronismo

(i.e. alguns processadores podem trabalhar com dados desatualizados). Consequentemente, quando um cache de CPU/Core carrega o TxRing para seu cache local, todos os outros processadores que também estiverem armazenando-o cache devem invalidar as cópias em seus caches. Assim, esta invalidação é muito custosa, pois dados compartilhados só podem ser salvos em cache por uma CPU/core de cada vez, mas também isso força os dados a serem carreados da RAM toda vez que muda a CPU/core processando. Isso introduz um *overhead* de acesso de memória que deve ser levado em conta.

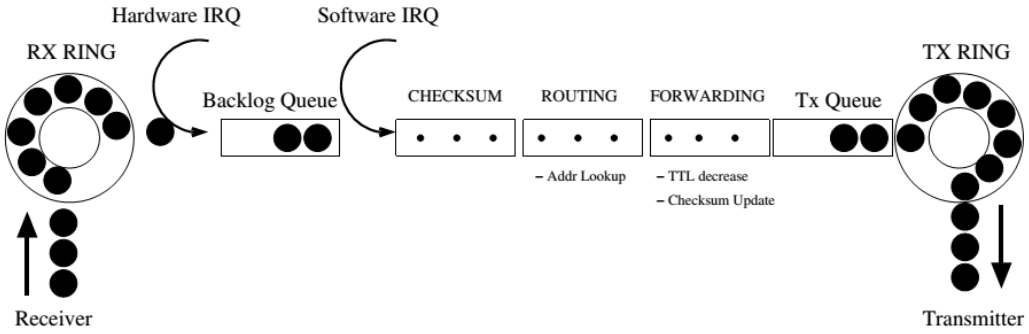

Figura 5 - plano de encaminhamento

2.11 QUAGGA

Quagga é um pacote de software para roteamento que fornece serviços de roteamento baseado em TCP/IP com suporte a protocolos de roteamento como RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, IS-IS, BGP-4 e BGP-4+.

Além do roteamento tradicional IPv4, o Quagga também suporta prototcolos de roteamento IPv6, usando uma arquitetura de SW para fornecer roteamento multi-servidor de alta qualidade. O Quagga tem uma interface de usuário interativa para cada protocolo de roteamento e suporta comandos de cliente comuns. Com este projeto, é possível novos deamons (pequenos programas que rodam em plano de fundo) de protocolos facilmente.

O Quagga é distribuído sob a gnu General Public License. Um sistema com Quagga instalado age como um roteador dedicado, assim o computador troca informações de roteamento com outros roteadores usando protocolos de

roteamento. Desta forma o Quagga atualiza a tabela de roteamento do kernel para que os dados corretos vão aos locais corretos. (ISHIGURO & al., 2013)

É possível alterar dinamicamente a configuração e visualizar a tabela de roteamento da interface de terminal do Quagga.

Tradicionalmente, a configuração de roteadores baseados em Linux é feita pelos comandos “ifconfig” e “route”. O status da tabela de roteamento é mostrado com comando “netstat” mas estes comandos só funcionam se o usuário possuir privilégios de root. O Quagga tem um método de administração de sistema diferente. Há dois modos de usuário no Quagga: normal e enable. No modo normal só permite visualizar status do sistema, já o modo enable pode modificar configurações. Esta independência dos privilégios de usuário do Unix é de grande ajuda para o administrador do roteador.

2.11.1 Arquitetura do Sistema

Roteamento por software tradicionalmente é feito como programa que fornece todas as funcionalidades de protocolo de roteamento. O Quagga tem uma abordagem diferente, feita de uma coleção de vários daemons que trabalham juntos para construir a tabela de roteamento. Podem haver vários daemons de protocolos de roteamento específicos e o gerenciador de roteamento do kernel “Zebra”, como mostrado na figura:

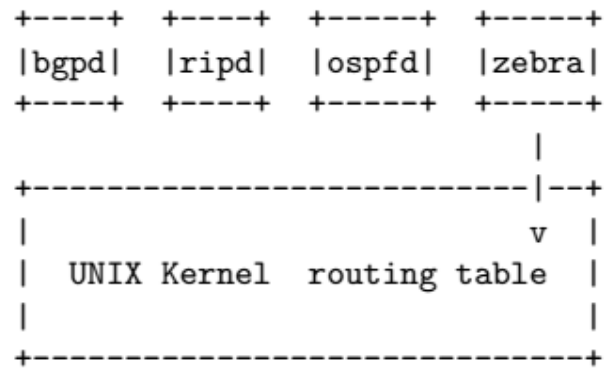

Figura 6 - Relação entre daemons Quagga e o Unix/Linux

O daemon “ripd” manipula o protocolo RIP, enquanto o “ospfd” é o daemon que suporta o protocolo OSPFv2. “bgpd” suporta o protocolo BGP-4. Para alterar a

tabela de roteamento do kernel e para redistribuição de rotas entre diferentes protocolos de roteamento, há o daemon “zebra” que é o gerenciador da tabela de roteamento do kernel.

É possível adicionar novos daemons de protocolo de roteamento ao sistema como um todo sem afetar qualquer outro software do Linux. Para isso é necessário apenas rodar o daemon do protocolo associado com protocolos de roteamento em uso. Assim, o usuário pode rodar um daemon específico e enviar reports de roteamento a um console de roteamento central.

Arquitetura multi-processos traz capacidade de extensão, modularidade e habilidade de manutenção. Ao mesmo tempo também traz muitos arquivos de configuração e interfaces de terminal. Cada daemon tem seu próprio arquivo de configuração e interface de terminal. Quando é configurada uma rota estática, é necessário que seja no arquivo de configuração “zebra”. Para configurar uma rede BGP isto deve ser feito no arquivo de configuração “bgpd”. Isso pode ser uma coisa enfadonha, então para resolver esta questão o Quagga fornece uma interface de usuário shell integrada em chamada “vtyh” que se conecta a cada daemon e age como um proxy para input de usuário.

Atualmente o Quagga supoerta plataformas gnu/Linux e BSD, enquanto os daemons de protocolos são na maioria independentes de plataforma. (ISHIGURO & al., 2013)

3 DESENVOLVIMENTO

Neste capítulo será descrito o cenário para testes em campo, equipamentos utilizados, scripts rodados e testes executados para validação.

3.1 PROCEDIMENTOS

Primeiramente foi projetada a rede de teste com simulador de redes Cisco Packet Tracer v6.1 Student. Como o objetivo é o teste de conectividade de camada IP entre as diferentes redes, foi usado apenas um host em cada rede. Desta forma a topologia adquiriu o seguinte formato:

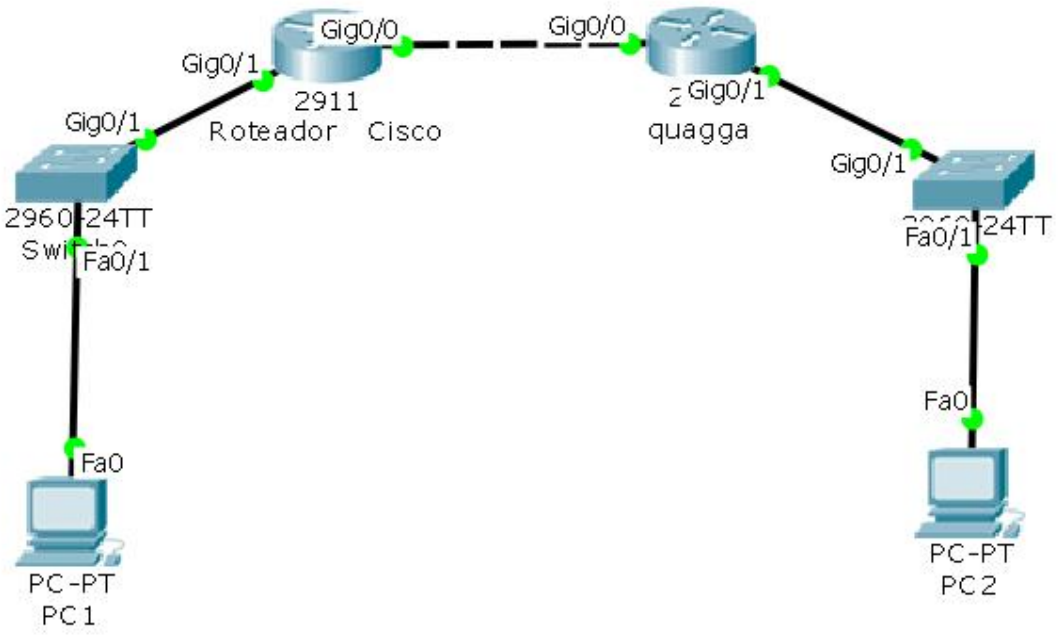

Figura 7 - Topologia de rede a ser testada

A seguir, os endereços configurados, que serão usados também na prática em laboratório:

Interface	Roteador Cisco	Roteador Quagga
Gig 0/0	2001:db8:1:1::1/64	2001:db8:1:1::2/64
Gig 0/1	2001:db8:1:2::1/64	2001:db8:1:3::1/64

Tabela 2 - Endereços IPv6 dos Roteadores

	PC 1	PC 2
Endereço IPv6	2001:DB8:1:2::5/64	2001:DB8:1:3::5/64
Default Gateway	2001:DB8:1:2::1	2001:DB8:1:3::1

Tabela 3 - Endereços IPv6 dos Hosts no ambiente simulado Packet Tracer.

Abaixo segue o script de comandos elaborado para testes, carregado nos roteadores no ambiente simulado Packet Tracer. Estes comandos serão melhor descritos na seção em que falaremos dos testes reais.

ROTEADOR CISCO:

Configurando acesso ao roteador:

```
enable
conf term
hostname ciscorouter
enable password cisco
line vty 0 4
password cisco
login
exit
```

Configurando o roteamento IPv6 com OSPF:

```
ipv6 unicast-routing
ipv6 router ospf 1
router-id 1.1.1.1
passive-interface gi0/1
```

Configurando as interfaces:

```
interface gi0/0
ipv6 address 2001:db8:1:1::1/64
ipv6 ospf 1 area 0
no shut

interface gi0/1
```

```
ipv6 address 2001:db8:1:2::1/64
ipv6 ospf 1 area 1
no shut

end
wr
```

ROTEADOR QUAGGA:

Configurando acesso ao roteador:

```
enable
conf term
hostname quaggarouter
enable password cisco
line vty 0 4
password cisco
login
exit
```

Configurando o roteamento IPv6 com OSPF:

```
ipv6 unicast-routing
ipv6 router ospf 1
router-id 2.2.2.2
passive-interface gi0/1
```

Configurando as interfaces:

```
interface gi0/0
ipv6 address 2001:db8:1:1::2/64
ipv6 ospf 1 area 0
no shut
exit

interface gi0/1
ipv6 address 2001:db8:1:3::1/64
ipv6 ospf 1 area 1
no shut

end
wr
```

Verificando as configurações dos roteadores:

```
Show running config
show ipv6 route
show ipv6 interface brief
show ipv6 neighbors
show ipv6 protocols
show ipv6 traffic
show ipv6 ospf interface g0/0
show ipv6 ospf database
show ipv6 traffic
```

Resultados dos testes em ambiente simulado:

Para validação da conectividade entre as redes, o comando Ping deve ser executado de cada um dos PCs da rede, direcionado ao PC da outra rede. Sendo o resultado do comando bem-sucedido, significa que as redes estão configuradas e permitindo conexão de pacotes IPv6. Foi o que ocorreu, como pode ser verificado nas imagens a seguir.

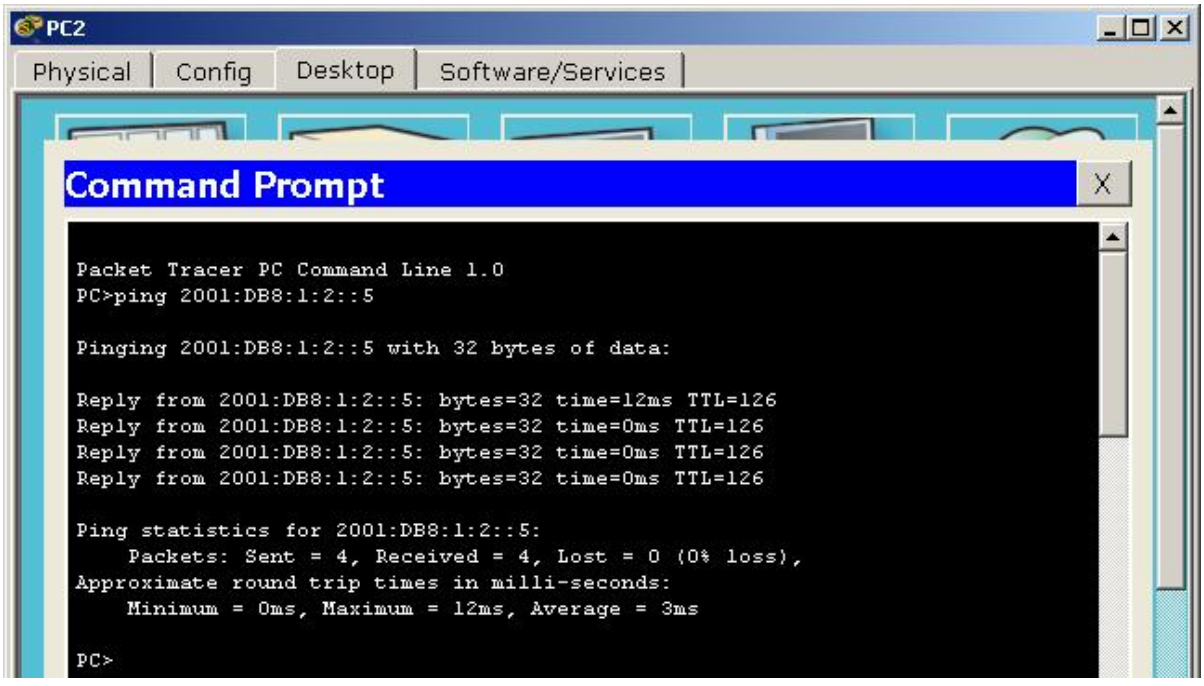

Figura 8 - Ping bem-sucedido do PC2 para o PC1

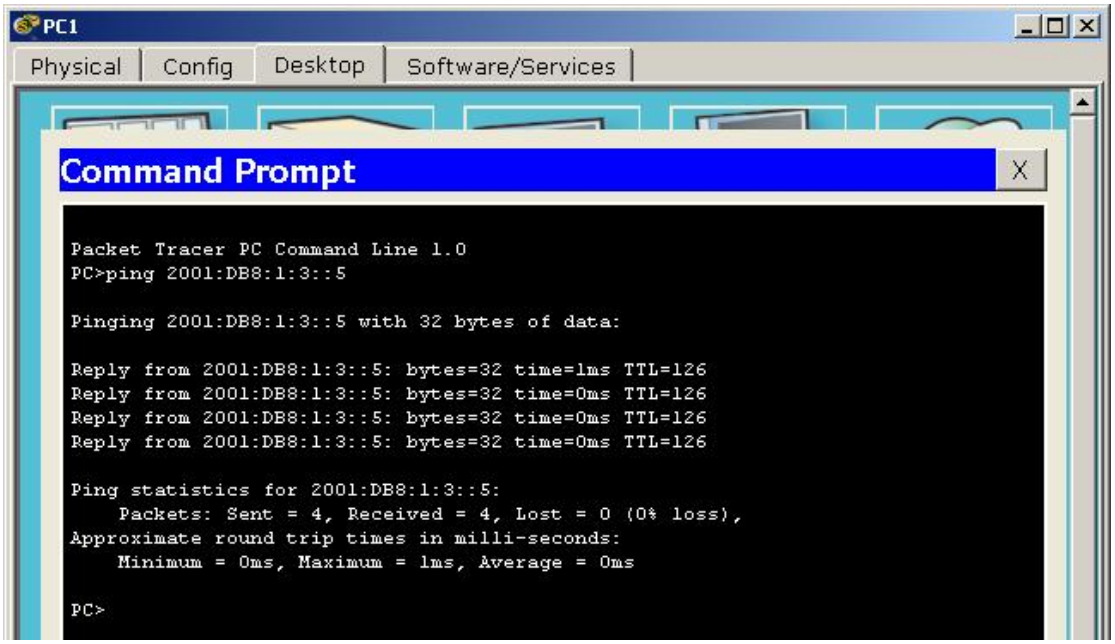

Figura 9 – Ping bem-sucedido do PC1 para o PC2

Após a validação bem-sucedida da topologia e configuração de roteadores em amiente de simulação, chegou a hora de entrar no laboratório e iniciar a parte prática.

MONTAGEM E CONFIGURAÇÃO DOS EQUIPAMENTOS EM AMBIENTE REAL:

Os testes de laboratório foram executados dentro do laboratório de rede do Senai (Serviço Nacional de Aprendizagem Industrial) em Curitiba, na Av. das Torres, 640. O material usado foi o seguinte:

- Um Roteador Cisco 2901 como o da Figura 10;
- Um Switch Cisco 2950;
- Um PC com duas placas de rede, executando sistema operacional Debian Wheezy 7.1 com Kernel Linux versão 3.2, usado como Roteador Quagga;
- Cabos de rede para conexão dos elementos;
- Um notebook para conexão de Console direta com o roteador;

Figura 10 - Roteador Cisco 2901

Fotos da bancada de equipamentos do laboratório:

Figura 11 - Switch e Roteador Cisco na bancada do Laboratório

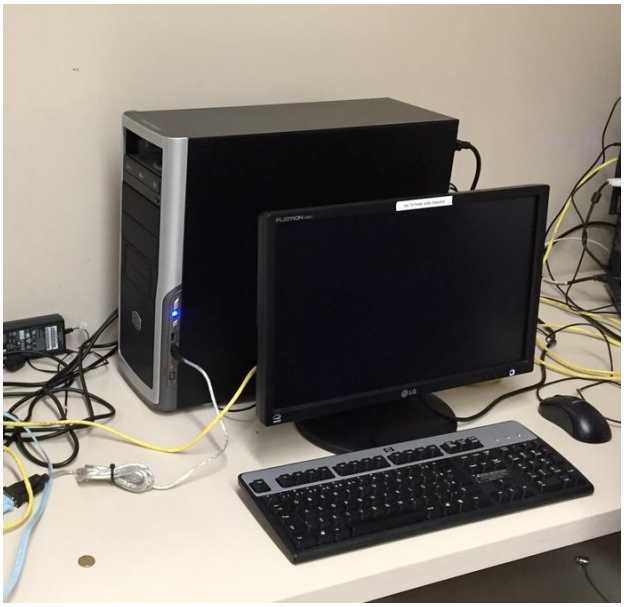

Figura 12 - PC usado como Roteador Quagga

O equipamento do laboratório foi conectado conforme a topologia apresentada para o cenário de simulação.

CONFIGURAÇÃO DO LINUX:

Antes de iniciar a rodar o script do roteador Quagga, é necessário preparar o PC Linux e também o próprio programa Quagga depois de instalado. A preparação do Linux não é tão trivial e deve obedecer alguns passos com detalhes importantes para habilitar o encaminhamento de pacotes e o endereçamento IPv6, preferencialmente antes da instalação do Quagga.

Primeiramente, vamos enviar os comandos abaixo para habilitar o encaminhamento de pacotes IP6 no Debian Linux. **Importante**: toda configuração do Linux deverá ser feita em modo Root (Super Usuário Linux).

Habilitando o encaminhamento de pacotes IPv6 no Debian:

```
# echo net.ipv6.conf.all.forwarding = 1 >>
/etc/sysctl.d/ipv6_forwarding.conf
# sysctl –p
```

Seguindo, vamos configurar as interfaces para receber os endereços IPv6. Para isto, é necessário editar o arquivo `/etc/network/interfaces` conforme segue:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet6 static
address 2001:db8:1:1::2
netmask 64

auto eth1
iface eth1 inet6 static
address 2001:db8:1:3::1
netmask 64
```

Nota: é possível configurar as interfaces também pelo comando ifconfig do Linux, ou então pela própria interface de console do Quagga. Porém, as configurações de IPv6 somente permanecerão após um reboot quando estiverem escritas no arquivo `interfaces`.

Para fazer o download e instalar o pacote Quagga:

```
# apt-get install quagga
```

DEAMONS QUAGGA:

É necessário ativar os daemons do Quagga correspondentes aos protocolos de roteamento a serem usados no roteador.

zebra:	Declaração de interfaces e roteamento estático
bgpd:	Protocolo de Roteamento BGP
ospfd:	Protocolo de Roteamento OSPF
ospf6d:	Protocolo de Roteamento OSPFv3 (para IPv6)
ripd:	Protocolo de Roteamento RIPv2
ripngd:	Protocolo de Roteamento RIPng (Para IPv6)

Tabela 4- Daemons do Quagga

O Daemon `zebra` deverá sempre estar ativo para o correto funcionamento do Quagga. Portanto em nosso caso habilitaremos apenas os daemons `zebra` e `ospf6d`, editando o arquivo `/etc/quagga/daemons` conforme a seguir:

```
zebra=yes
bgpd=no
ospfd=no
ospf6d=yes
ripd=no
ripngd=no
```

Após a configuração dos Daemons é necessário restart do Quagga, com o seguinte comando:

```
#/etc/init.d/quagga restart
```

Para checar o status dos daemons:

```
#ps -ef | grep quagga
```

ARQUIVOS DE CONFIGURAÇÃO DO QUAGGA (arquivos no diretório `/etc/quagga/*.conf`):

É necessário criar um arquivo de configuração (mesmo que vazio) para cada daemon ativado. Cada daemon é associado a um arquivo específico:

zebra: `zebra.conf`
bgpd: `bgpd.conf`

ospfd: `ospfd.conf`
ospf6d: `ospf6d.conf`
ripd: `ripd.conf`
ripngd: `ripngd.conf`

Uma forma de criar os arquivos .conf é copiar os arquivos de exemplo (criados na instalação do Quagga). Para este trabalho, é necessário criar os arquivos de configuração `zebra.conf` e `ospf6d.conf` conforme os comandos:

```
#cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf
#cp /usr/share/doc/quagga/examples/ospf6d.conf.sample
/etc/quagga/ospf6d.conf
```

Finalmente, dê posse e permissão dos arquivos dentro do diretório `/etc/quagga` ao usuário *quagga* e grupo *quaggavty* com os comandos:

```
#chown quagga.quaggavty /etc/quagga/*.conf
#chmod 640 /etc/quagga/*.conf
```

Restart no Quagga:

```
#/etc/init.d/quagga restart
```

ARQUIVO `DEBIAN.CONF`:

Por padrão, os daemons Quagga só podem ser acessados pela interface loopback (127.00.1). Para acessar remotamente os daemons via telnet, é necessário indicar um ou vários endereços IP no arquivo `/etc/quagga/debian.conf` ou remover a opção –A significando que você pode fazer telnet em um daemon em qualquer dos seus endereços IP. Seguem dois exemplos:

O daemon ospf6d está “escutando” os endereços IP 127.0.0.1 e 192.168.1.104:

```
ospf6d_options=" --daemon -A 127.0.0.1 192.168.1.104"
```

O daemon zebra está escutando todos os endereços IP das interfaces Linux. É recomendável usar esta opção.

```
zebra_options=" --daemon "
```

Se você quiser filtrar quem pode acessar seu roteador, configure ACL (Access Control List, lista de controle de acesso) no Linux.

Este é o arquivo `debian.conf` como ficou após a configuração prática em laboratório:

```
# If this option is set the /etc/init.d/quagga script automatically
loads
# the config via "vtysh -b" when the servers are started.
# Check /etc/pam.d/quagga if you intend to use "vtysh"!
#
vtysh_enable=yes
zebra_options="  --daemon -A 127.0.0.1"
bgpd_options="   --daemon -A 127.0.0.1"
ospfd_options="  --daemon -A 127.0.0.1"
ospf6d_options=" --daemon -A ::1"
ripd_options="   --daemon -A 127.0.0.1"
ripngd_options=" --daemon -A ::1"
isisd_options="  --daemon -A 127.0.0.1"
babeld_options=" --daemon -A 127.0.0.1"
#
# Please note that watchquagga_options is an array and not a string
so that
# quotes can be used.
#
# The list of daemons to watch is automatically generated by the
init script
# from daemons.conf and appended to the watchquagga_options.
# Example:
#    watchquagga_options=("-Adz" "-r" '/sbin/service %s restart' -s
'/sbin/service %s start'  -k '/sbin/service %s stop')
watchquagga_enable=yes
watchquagga_options=(--daemon)
```

Restart no Quagga:

```
#/etc/init.d/quagga restart
```

VTYSH:

O vtysh foi criado para configuração de roteamento do Quagga em uma única interface, ao invés de acessar cada daemon separadamente com telnet. Para usar o vtysh, é necessário primeiro criar seu arquivo de configuração conforme a seguir:

```
#cp /usr/share/doc/quagga/examples/vtysh.conf.sample /etc/quagga/vtysh.conf
```

Aplique novamente as permissões corretas e faça o restart do Quagga:

```
#chown quagga.quaggavty /etc/quagga/*.conf
#chmod 640 /etc/quagga/*.conf
#/etc/init.d/quagga restart
```

Neste arquivo `vtysh.conf` é possível definir como serão salvas as configurações do Quagga. Para salvar todas configurações em arquivos separados, é necessário comentar a linha "*service integrated-vtysh-config*", assim as configurações serão salvas nos arquivos zebra.conf e nos arquivos .conf dos daemons ativados. Em nosso estudo, esta linha está ativa e portanto a configuração do roteador está salva no arquivo /etc/quagga/Quagga.conf.

Finalmente agora o Linux está preparado para receber as configurações do Quagga. Vamos aos comandos para configuração do Roteador Quagga (lembrando: as interfaces já foram configuradas no nível Linux, portanto não serão alteradas dentro do vtysh.

Entrando na interface do Quagga pelo Linux, em modo root:

```
#vtysh
```

Script para ativação do OSPFv3 no Quagga:

```
enable
conf term
hostname quaggarouter
exit
```

```
ipv6 router ospf 1
router-id 2.2.2.2
exit

int eth0
ipv6 ospf 1 area 0
int eth1
ipv6 ospf 1 area 1

end
wr
```

CONFIGURAÇÃO DO ROTEADOR CISCO:

```
enable
conf term
hostname ciscorouter

enable password cisco
line vty 0 4
password cisco
login
exit

‘ativando roteamento IPv6
ipv6 unicast-routing

‘ativando roteamento OSPFv3
ipv6 router rip quagga

'interface para o quagga
interface gi0/0
ipv6 address 2001:db8:1:1::1/64
ipv6 rip quagga enable
no shut

'interface lan
interface gi0/1
ipv6 address 2001:db8:1:2::1/64
ipv6 rip quagga enable
```

```
no shut
```

CONFIGURAÇÃO DOS PCs USADOS NA PRÁTICA:

Foi mantido o endereçamento IP da Tabela 3. O PC1 na prática não foi implementado, mas foram usadas interfaces loopback no roteador Cisco para testes de Ping a partir do PC2.

3.2. RESULTADOS

Após todas as configurações do Linux e carregamento dos scripts para o teste prático nos roteadores, temos os resultados a seguir. Começando com a verificação da interface Gi0/0 do roteador Cisco com o roteador Quagga em estado ativo (up/up) e endereço IP configurado.

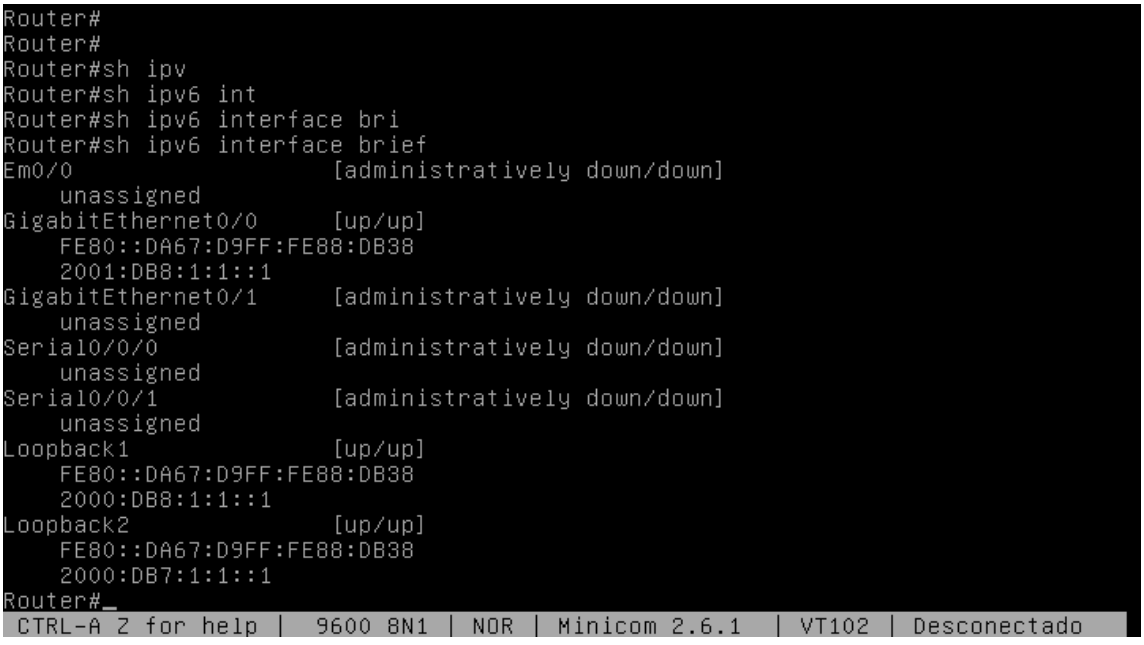

Figura 13 – Interfaces do IPv6 no roteador Cisco.

Após aguardar a rápida convergência do protocolo OSPFv3, foi possível verificar nas tabelas de roteamento o anúncio das redes. Na Figura 14, verifica-se que o protocolo convergiu, e o roteador Cisco adquiriu via OSPFv3 a rede

2001:db8:1:3::/64 conectada ao roteador Quagga, que é a rede onde o PC2 está conectado.

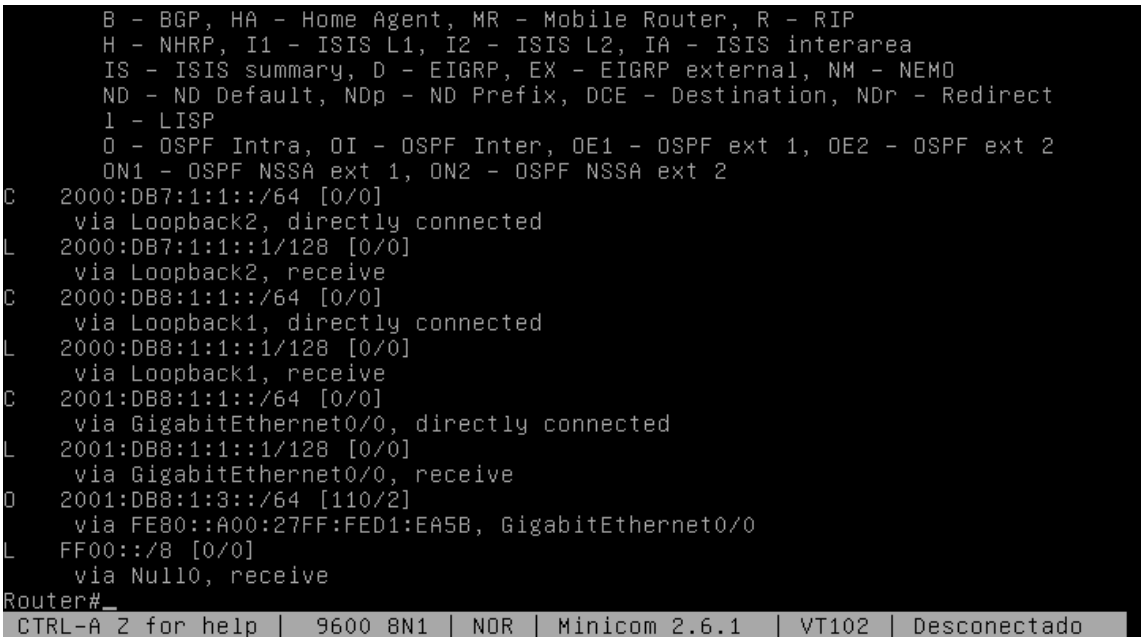

Figura 14 - Tabela de roteamento do Roteador Cisco contendo anúncio OSPFv3 para rede do roteador Quagga

Do mesmo modo, o roteador Quagga também adquiriu via OSPFv3 as redes Loopback do roteador Cisco, como mostrado na Figura 15.

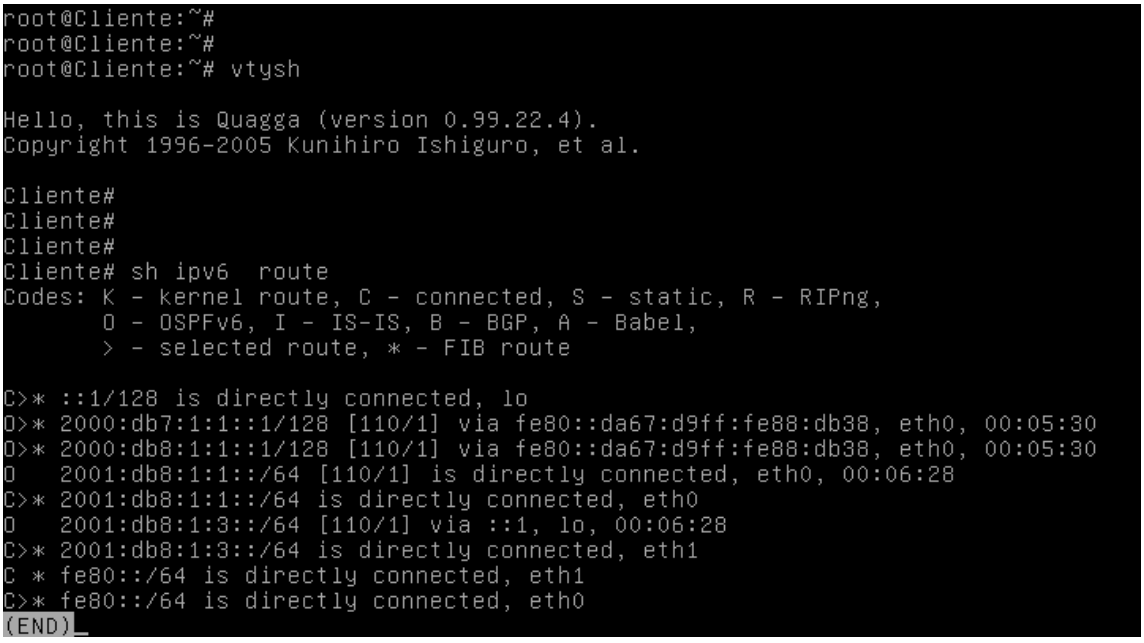

Figura 15 - Tabela de roteamento do Roteador Quagga contendo anúncios OSPFv3 das redes Loopback do roteador Cisco.

Tendo as redes devidamente anunciadas entre os roteadores e suas tabelas de roteamento atualizadas, foi possível avançar e executar os testes de conectividade entre as redes com comando Ping. Seguem abaixo os resultados do comando Ping.

A Figura 16 mostra o PC2 executando Ping com sucesso para a interface do Roteador Cisco e interfaces loopback criadas para teste:

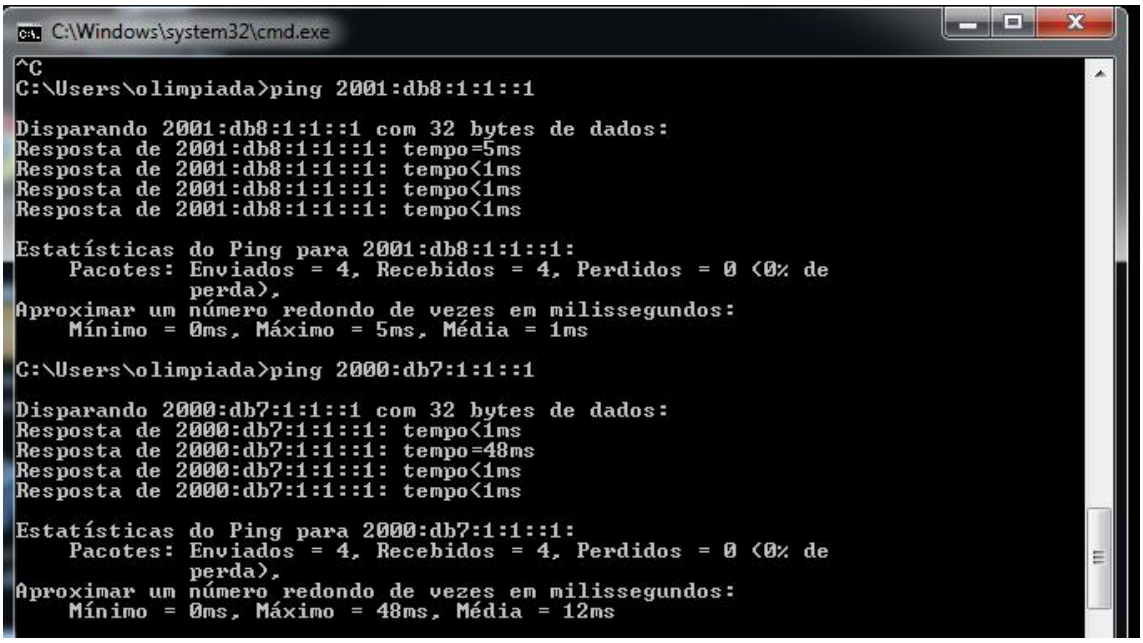

Figura 16 - ping a partir do PC2 para a interface do roteador Cisco e sua interface loopback

Como última informação da prática, segue o conteúdo do arquivo de configuração `/etc/quagga/Quagga.conf`, equivalente ao comando `show running config` do roteador Cisco.

```
hostname Router
hostname ripngd
log stdout
hostname ospfd
hostname ospf6d@plant
!
service advanced-vty
!
debug ospf6 lsa unknown
debug ospf6 neighbor state
!
password zebra
enable password zebra
!
interface eth0
```

```
 ipv6 nd suppress-ra
!
interface eth1
 ipv6 nd suppress-ra
!
interface eth2
 ipv6 nd suppress-ra
!
interface lo
!
interface fxp0
 ipv6 ospf6 priority 0
!
interface lo0
!
router ripng
!
router ospf6
 router-id 2.2.2.2
 redistribute static route-map static-ospf6
 interface fxp0 area 0.0.0.0
 interface eth0 area 0.0.0.0
!
access-list access4 permit 127.0.0.1/32
!
ipv6 access-list access6 permit 3ffe:501::/32
ipv6 access-list access6 permit 2001:200::/48
ipv6 access-list access6 permit ::1/128
!
ipv6 prefix-list test-prefix seq 1000 deny any
!
route-map static-ospf6 permit 10
 match ipv6 address prefix-list test-prefix
 set metric 2000
 set metric-type type-2
!
ip forwarding
ipv6 forwarding
!
line vty
 access-class access4
 exec-timeout 0 0
 ipv6 access-class access6
!
```

3.3. CONCLUSÃO E FUTURO TRABALHO

Esta prática provou que é possível interconectar redes entre fabricantes distintos utilizando protocolos padronizados por órgãos internacionais como é o caso do OSPFv3, definido pela IETF (Internet Engineering Task Force) através da RFC

5340. Isto ajuda a implementar redes de forma autônoma, sem depender de protocolos específicos e proprietários de fabricantes.

A convergência da rede ocorreu de forma muito rápida, característica do protocolo OSPFv3 conforme já mencionado. O tutorial de instalação criado no Item 3.1 evidencia uma experiência de

Como tópicos a serem explorados ainda dentro desta linha de estudo, é recomendável a implementação de requisitos de Segurança de Rede como Firewall, ACLs e senha de acesso ao próprio roteador. Além disso, caso seja implementado VoIP na rede, também é interessante verificar a questão do QoS.

A questão do desempenho da rede não foi objeto de estudo deste trabalho, mas certamente a capacidade de roteamento do Linux - sendo um HW de propósito geral – não será tão grande quanto a do roteador Cisco, que serve especificamente o propósito de rotear pacotes IP entre redes.

4. REFERÊNCIAS

BIANCO, A., FINOCHIETTO, J., GALANTE, G., MELLIA, M., & NERI, F. (Fevereiro de 2005). Open-Source PC-Based Software Routers: A ViableApproach to High-Performance Packet Switching. *Quality of Service in Multiservice IP Networks*, pp. 353–366.

BOLLA, R., & BRUSCHI, R. (22 de Agosto de 2008). PC-based Software Routers: High Performance and Application Service Support. *PRESTO*, pp. 27-32.

Cisco Systems, Inc. (2009). CCNA 4.0 Exploration. Fonte: http://www.netacad.com

HUNT, C. (2002). *TCP/IP Network Administration.* O Reilly.

ISHIGURO, K., & al., e. (Janeiro de 2013). *Documentation.* Acesso em 12 de Outubro de 2014, disponível em Qagga Software Routing Suite: http://www.nongnu.org/quagga/docs/docs-info.html

Mims, C. (23 de Agosto de 2010). *Software Router Smashes Speed Records*. Acesso em 5 de Outubro de 2014, disponível em MIT Technology Review: http://www.technologyreview.com/news/420324/software-router-smashes-speed-records/

SIYAN, K., & PARKER, T. (2002). *TCP/IP Unleashed.* SAMS.