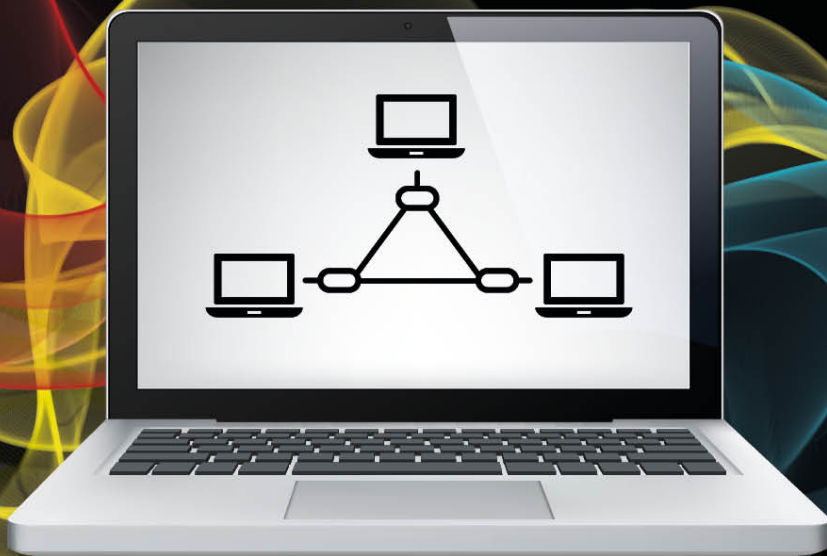




Laboratório de IPv6



Aprenda na prática usando
um emulador de redes

novatec

Equipe **IPv6.br**

Laboratório de **IPv6**

Aprenda na prática usando
um emulador de redes

Licença

Este livro está sob a licença Creative Commons Atribuição – Uso não Comercial – Com compartilhamento pela mesma licença 4.0 Internacional (CC BY-NC-SA 4.0), que está aqui resumida e pode ser lida em sua íntegra em: <http://creativecommons.org/licenses/by-nc-sa/4.0/> **Isso significa que você pode copiar partes ou todo este livro e redistribuir o material em qualquer suporte ou formato. Você pode também modificar o material da forma que desejar.** Você deve, contudo, atribuir o devido crédito, informando que o livro original pode ser obtido no *site* <http://lab.ipv6.br>. Você não pode utilizar este livro para fins comerciais e obras derivadas devem seguir esta mesma licença.



Dados dos Internacionais de Catalogação na Publicação (CIP) (Câmara Brasileira do Livro, SP, Brasil)

Laboratório de IPv6 [livro eletrônico] : aprenda na prática usando um emulador de redes / Equipe IPV6.br. – São Paulo : Novatec Editora, 2015.
11,6 Mb ; PDF.

Vários autores.
Bibliografia.
ISBN 978-85-7522-434-2

1. Internet (Rede de computadores) 2. Redes de computadores - Protocolos I. Equipe IPV6.br.

15-04022 CDD-004.62

Índices para catálogo sistemático:

1. Redes de computadores : Protocolos e aplicações : Processamento de dados 004.62

Laboratório de IPv6

Aprenda na prática usando
um emulador de redes

Equipe **IPv6.br**

Antonio Marcos Moreiras
Rodrigo Regis dos Santos
Alexandre Yukio Harano
Edwin Santos Cordeiro
Tiago Jun Nakamura
Eduardo Barasal Morales
Heitor de Souza Ganzeli
Rodrigo Matos Carnier
Gustavo Borges Lugoboni

Novatec

Esta é uma publicação do:

Núcleo de Informação e Coordenação do Ponto BR – NIC.br

Diretor Presidente: Demi Getschko

Diretor Administrativo: Ricardo Narchi

Diretor de Serviços: Frederico Neves

Diretor de Projetos Especiais e de Desenvolvimento: Milton Karu Kashiwakura

Diretor de Assessoria às Atividades do CGI.br: Hartmut Richard Glaser

Coordenação Executiva e Editorial: Antonio Marcos Moreiras

Revisão: Rodrigo Regis dos Santos, Alexandre Yukio Harano, Tiago Jun Nakamura, Tuany Oguro Tabosa e Alexei Dimitri Diniz Campos

Editoração Eletrônica: Alexandre Yukio Harano e Gustavo Borges Lugoboni

Capa: Maricy Rabelo

Sobre o CEPTR0.br e sobre o projeto IPv6.br

O Centro de Estudos e Pesquisas em Tecnologias de Redes e Operações é a área do NIC.br responsável por iniciativas que visam melhorar a qualidade da Internet no Brasil e disseminar seu uso, com especial atenção para seus aspectos técnicos e de infraestrutura. A equipe do CEPTR0 desenvolve soluções em infraestrutura de redes, *software* e *hardware*, além de gerenciar projetos executados por parceiros externos.

Um dos principais projetos do CEPTR0 é o IPv6.br, que engloba uma série de iniciativas do NIC.br para disseminar o IPv6 no Brasil. Entre elas, este livro.

O IPv6.br oferece cursos presenciais gratuitos, com teoria e prática, para provedores Internet e outras instituições. Entre 2009 e 2013, mais de 3000 pessoas foram capacitadas nesses treinamentos. Os experimentos para aprendizado do IPv6 apresentados neste trabalho foram criados para uso nesses cursos e isso vem sendo feito com ótimos resultados.

As iniciativas englobam ainda a realização de reuniões de coordenação com diversas entidades, visando a estratégia para implantação do IPv6 no país; a disponibilização de informações e de material didático no *site* <http://ipv6.br>, cursos em formato e-learning e EaD, palestras em universidades, empresas e eventos de tecnologia; bem como a realização de eventos sobre o IPv6, como os “Fóruns Brasileiros de IPv6” e os diversos “IPv6 no Café da Manhã”.

Sobre os autores

Este livro foi escrito a muitas mãos pela equipe do IPv6.br, do Centro de Estudos e Pesquisas em Tecnologias de Redes e Operações, do NIC.br. A equipe é formada por engenheiros, analistas e estudantes com conhecimentos e experiência em desenvolvimento de *software*, no funcionamento da Internet e das redes de computadores.

Sumário

Lista de abreviaturas	iii
Prefácio	v
Introdução	1
1 Funcionalidades básicas	9
Experiência 1.1 NDP: <i>Neighbor Solicitation</i> e <i>Neighbor Advertisement</i>	9
Experiência 1.2 NDP: <i>Router Solicitation</i>	15
Experiência 1.3 NDP: <i>Router Advertisement</i>	22
Experiência 1.4 NDP: detecção de endereços duplicados	27
Experiência 1.5 SLAAC: <i>Router Advertisement</i> utilizando Quagga	35
Experiência 1.6 SLAAC: <i>Router Advertisement</i> utilizando radvd	41
Experiência 1.7 DHCPv6 <i>stateful</i>	51
Experiência 1.8 DHCPv6 <i>stateless</i>	67
Experiência 1.9 DHCPv6 <i>Prefix Delegation</i>	81
Experiência 1.10 <i>Path MTU Discovery</i>	99
2 Serviços	105
Experiência 2.1 DNS: consultas DNS	105
Experiência 2.2 DNS: servidor autoritativo	117
Experiência 2.3 HTTP: novas páginas no Apache	130
Experiência 2.4 HTTP: páginas existentes no Apache	135
Experiência 2.5 HTTP: novas páginas no Nginx	141

Experiência 2.6	<i>Proxy Web</i> direto	147
Experiência 2.7	<i>Proxy Web</i> reverso	160
Experiência 2.8	Samba	168
3	Segurança	177
Experiência 3.1	Ataque DoS ao NDP	177
Experiência 3.2	<i>Firewall stateful</i>	185
Experiência 3.3	IPsec: Transporte	217
Experiência 3.4	IPsec: Túnel	244
4	Técnicas de transição	255
Experiência 4.1	Túnel 6in4	255
Experiência 4.2	Túnel GRE	262
Experiência 4.3	Dual Stack Lite (DS-Lite): implantação	268
Experiência 4.4	6rd: configuração de relay e CPE (/64)	276
Experiência 4.5	6rd: configuração de CPE (/56)	285
Experiência 4.6	NAT64: implantação utilizando TAYGA	293
Experiência 4.7	464XLAT	310
5	Roteamento	323
Experiência 5.1	OSPFv3: configuração de uma única área	323
Experiência 5.2	BGP	334
A	Instalação de pacotes	363
B	Emulador de redes CORE	367
C	Comandos básicos	379
	Referências Bibliográficas	389
	Índice remissivo	393

Lista de abreviaturas

6rd	<i>IPv6 Rapid Deployment on IPv4 Infrastructures</i>
A+P	<i>Address plus Port</i>
ARP	<i>Address Resolution Protocol</i>
CPE	<i>Customer Premises Equipment</i>
DAD	<i>Duplicate Address Detection</i>
DHCP	<i>Dynamic Host Configuration Protocol</i>
DNS	<i>Domain Name System</i>
DNS64	<i>DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers</i>
DoS	<i>Denial-of-Service</i>
DS-Lite	<i>Dual-Stack Lite</i>
IA	<i>Identity-Association</i>
ICMP	<i>Internet Control Message Protocol</i>
ICMPv4	<i>Internet Control Message Protocol version 4</i>
ICMPv6	<i>Internet Control Message Protocol version 6</i>
ICP	<i>Internet Cache Protocol</i>
HTTP	<i>Hypertext Transfer Protocol</i>
IGMP	<i>Internet Group Management Protocol</i>
IGP	<i>Interior Gateway Protocol</i>
IP	<i>Internet Protocol</i>
IPsec	<i>Internet Protocol Security</i>

IPv4	<i>Internet Protocol version 4</i>
IPv6	<i>Internet Protocol version 6</i>
ISO	<i>International Organization for Standardization</i>
ISP	<i>Internet Service Provider</i>
MAC	<i>Media Access Control</i>
MTU	<i>Maximum Transmission Unit</i>
NA	<i>Neighbor Advertisement</i>
NAT	<i>Network Address Translator</i>
NAT64	<i>Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers</i>
NDP	<i>Neighbor Discovery Protocol</i>
NS	<i>Neighbor Solicitation</i>
OSI	<i>Open Systems Interconnection</i>
PMTUD	<i>Path MTU Discovery</i>
RA	<i>Router Advertisement</i>
RARP	<i>Reverse Address Resolution Protocol</i>
RR	<i>Resource Record</i>
RS	<i>Router Solicitation</i>
SA	<i>Security Association</i>
SAD	<i>Security Association Database</i>
SLAAC	<i>IPv6 Stateless Address Autoconfiguration</i>
SEND	<i>SEcure Neighbor Discovery</i>
SPD	<i>Security Policy Database</i>
TTL	<i>Time To Live</i>
VM	<i>Virtual Machine</i>

Prefácio

A concepção da Internet é um dos marcos tecnológicos mais contundentes da sociedade moderna, tendo propiciado mudanças significativas de paradigmas do “como fazer” nas mais diversas áreas incluindo educação, entretenimento, pesquisa, transporte, comércio, saúde, entre diversas outras. Essa repercussão tão intensa e bem sucedida deve-se, em grande parte, à simplicidade dos seus protocolos de comunicação, mais especificamente de seus dois protocolos principais: o TCP (*Transmission Control Protocol*) e o IPv4 (*Internet Protocol*). O IPv4 tem como função principal viabilizar a interconexão de redes, sendo responsável basicamente pelo endereçamento lógico neste ambiente, segmentação, priorização de pacotes e descarte de pacotes com problemas de roteamento. Considerando que a concepção da Internet data da década de 70 e que, de lá para cá, houve uma explosão inesperada do seu uso, o IPv4 mostrou-se inadequado para acompanhar esta evolução.

Uma das deficiências mais apontadas do IPv4 foi o espaço de endereçamento baseado num valor inteiro de 32 *bits*, que é tipicamente representado por quatro octetos em decimal. Para contornar essa deficiência, inúmeras soluções paliativas foram propostas e adotadas, como por exemplo o NAT (*Network Address Solution*) e o CIDR (*Classless InterDomain Routing*). Contudo, à medida que novas tecnologias de redes surgiram e o IP continuava sendo um dos protocolos chaves para sua operação, outras deficiências começaram a ser detectadas, especialmente aquelas referentes à segurança e ao suporte a parâmetros de QoS (*Quality of Service*) e mobilidade.

Como consequência, no início da década de 90 é publicada a proposta da nova geração do IP (IPng – IP *next generation*) ou IPv6. Este novo protocolo traz a solução para muitas das deficiências de seu predecessor, o IPv4, incluindo espaço de endereçamento de 128 *bits*, suporte a roteamento e segmentação de pacotes na estação origem, suporte a mobilidade e mecanismos de segurança.

Contudo, desde então, o IPv6 não foi amplamente adotado, apesar de esforços e incentivos de diversos governos, como o americano e brasileiro. Os principais motivos para resistência à sua adoção são o grande parque de equipamentos instalados com IPv4 nativo, os custos de implantação de soluções de migração (como por exemplo, *Dual Stack* ou Pilha Dupla) do IPv4 para IPv6 e a própria curva de aprendizado.

A despeito disso, a especificação do IPv6 tem sido continuamente revisada para acompanhar a evolução tecnológica das redes de computadores e sua crescente penetração nos mais diversos setores da economia. Dentre os vários avanços tecnológicos, podem-se mencionar a convergência de telefonia e redes de computadores; a mobilidade; mecanismos de segurança; a adoção crescente de mídias de alta resolução e a necessidade de seu compartilhamento; o advento de Internet das Coisas (*Internet of Things*) indo em direção à Internet de Tudo (*Internet of Everything*).

Tudo isso indica que o IPv4 tem seus dias contados e o IPv6 já está batendo na sua porta em virtude nesta nova realidade cada vez mais categórica de um mundo conectado dentro do contexto de Internet de Tudo (*Internet of Everything*). Essa realidade traz enormes desafios, como a necessidade do desenvolvimento de competências técnicas na área de IPv6.

Este livro, intitulado **Laboratório de IPv6: aprenda na prática usando um emulador de redes** ajuda a preencher a lacuna no ensino de redes de computadores e na criação de competência técnica no que tange ao IPv6. Mesmo que a padronização do protocolo já tenha acontecido há mais de quinze anos, o IPv6 foi relegado a um segundo plano na formação dos profissionais, conforme explicado anteriormente. Se o seu estudo não o incluiu quando aprendeu a lidar com redes, este livro

o ajudará a reciclar seu conhecimento. Se você está aprendendo sobre redes agora e quer ter uma visão mais prática e experimental, ótimo. Você está no caminho certo. O IPv6 em substituição do IPv4, será, em breve, o protocolo mais utilizado em redes em geral.

Como o próprio nome indica, este livro tem um caráter prático e contém roteiros para experimentos que podem auxiliá-lo no seu aprendizado. Ele pode ser usado tanto por quem está começando a aprender sobre redes agora, como por profissionais experientes. Não é um livro apenas para ler, você deve realizar os experimentos. Também é importante entender que ele não é completo. Isto é, não aborda toda a teoria ou todos os tópicos necessários para uma compreensão completa de redes ou do IPv6. É um complemento. Para aproveitar bem este conteúdo, você deve ter ao menos lido sobre IPv6 e sobre redes em algum outro lugar. Caso ainda não tenha feito isto, pode acessar o *site* <http://ipv6.br> e fazer o curso *e-learning* gratuito ou ler o material teórico lá disponível.

A equipe do IPv6.br – o projeto de disseminação do IPv6 do NIC.br – foi quem preparou e aperfeiçoou estes experimentos. Tais experimentos tem sido empregados, com muito sucesso, nos cursos de formação do NIC.br. Centenas de alunos e de profissionais já seguiram estes mesmos roteiros. Eles comprovadamente ajudam a entender a forma como o IPv6 funciona, como se diferencia do protocolo IPv4, e como realizar configurações na prática em uma série de situações. Os experimentos proporcionam uma excelente base prática, que o ajudará muito no seu dia a dia.

Utiliza-se o emulador de redes CORE. É um ambiente gráfico, que permitirá a você experimentar diversas topologias e configurações de redes diferentes. No *site* do livro, <http://lab.ipv6.br>, você pode baixar uma imagem de máquina virtual, que funciona em qualquer sistema operacional com o VirtualBox. Recomenda-se seu uso.

O CORE roda nativamente no Linux ou no FreeBSD. Então, apesar de ser recomendado o uso de máquina virtual baseada no VirtualBox, o Apêndice A contém um guia para a instalação do emulador e outras dependências, caso você decida fazer isso diretamente em seu computador. Somente faça isso se tiver uma boa experiência.

No Apêndice B, você pode encontrar mais informações sobre o CORE e um guia básico sobre como utilizá-lo. Caso você nunca tenha usado esse emulador antes, revise este guia antes de se aventurar pelos experimentos.

No Apêndice C, foram incluídas algumas dicas sobre: como realizar tarefas comuns a quase todas as experiências; como verificar os endereços IP; testar a conectividade entre dispositivos, capturar e analisar pacotes. Caso você não conheça bem o ambiente Linux e não esteja acostumado com as ferramentas utilizadas nas experiências, este apêndice também deverá ser lido.

A ordem dos experimentos no livro é a mesma utilizada nos cursos do NIC.br. Caso seja novato no assunto e queira ter uma visão geral, você poderá realizar as experiências na ordem em que são apresentadas. Caso contrário, você poderá buscar as que mais se adequam às suas necessidades imediatas.

No capítulo 1, constam experimentos sobre o funcionamento básico do IPv6. É explicado: como ele faz o mapeamento com a camada de enlace; como funciona a autoconfiguração *stateless*, sem uso de DHCP (*Dynamic Host Configuration Protocol*); como usar o DHCPv6 e o *prefix delegation*; como é descoberto o valor da MTU (*Maximum Transmission Unit*). Tudo isto funciona de forma diferente do IPv4 e, se você quer entender realmente bem o IPv6 e ser capaz de resolver problemas em sua rede, estes experimentos irão ajudá-lo.

No capítulo 2, as experiências abordam serviços importantes em uma rede, como DNS (*Domain Name System*), servidor *Web*, *proxy* e servidor de arquivos. Você verá que as configurações não são muito diferentes das usadas com o protocolo antigo, mas que há alguns detalhes importantes a serem observados. Se você vai ativar o IPv6 nesses serviços, faça estes experimentos.

No capítulo 3, aborda-se a questão da segurança. Assim como o protocolo antigo, o IPv6 tem também vulnerabilidades. Uma das experiências ilustra isto, explorando uma falha no protocolo de descoberta de vizinhança. Mostra-se, também, como configurar um *firewall* IPv6, com atenção especial às regras referentes ao ICMPv6 (*Internet Control Message Protocol*). Há também experimentos com configuração de IPsec. São experimentos recomendados a todos.

No capítulo 4 são mostrados diversos tipos de técnicas de migração e transição do IPv4 para o IPv6. Dentre tais técnicas, podem-se citar: Túneis 6in4 e GRE (*Generic Routing Encapsulation*), que são úteis de forma geral. O *Dual Stack Lite*, 6rd, NAT64 e 464XLAT são técnicas que podem ser usadas por provedores de acesso à Internet.

O capítulo 5, por fim, aborda roteamento dinâmico, com um experimento sobre OSPFv3 e outro sobre BGP.

O principal objetivo da equipe do NIC.br ao desenvolver este livro foi criar uma referência que possa ajudá-lo a entender o IPv6 e usá-lo na prática do seu dia a dia. Caso tenha dúvidas, comentários ou sugestões, você pode entrar em contato pelo *site* <http://lab.ipv6.br>. O IPv6 será cada vez mais um tema obrigatório para profissionais experientes e iniciantes em redes. Bom aprendizado e muito sucesso!

Tereza Cristina Melo De Brito Carvalho

Prof^a. Associada da Escola Politécnica da USP

Coordenadora técnica de projetos do LARC-PCS-EPUSP

Introdução

Você já parou para pensar em quanto a Internet é útil em nosso dia a dia? Praticamente todo tipo de informação está disponível na rede. Advogados consultam o andamento de processos e atualizam-se sobre mudanças na legislação. Cidadãos relacionam-se com o Poder Público. Organizações não governamentais se articulam. Empresas oferecem serviços, compram e vendem por meio da Internet. Crianças e jovens jogam *online*. Donas de casa buscam entretenimento. Estudantes buscam conhecimentos sobre os mais variados assuntos, usam a rede para comunicar-se e trocar informações. Para os profissionais de computação é impensável ficar de fora: informações sobre linguagens de programação, novas técnicas, algoritmos, equipamentos, tutoriais, exemplos, o dia a dia está na rede. E tanto mais! A Internet já fez jovens com empresas criadas no fundo da garagem tornarem-se milionários e, de forma geral, ela tem ajudado no desenvolvimento pessoal de muita gente. Tem sido tão benéfica para os indivíduos e para a sociedade, que muito se discute hoje se o acesso à Internet não deveria ser considerado um direito fundamental do ser humano!

Isso foi sempre assim? Se pensarmos um pouco é bastante óbvio que não, pelo simples fato da Internet ser recente, extremamente recente. Por que a Internet é assim, hoje, tão útil para todos? Quais são as características que permitiram que ela evoluísse desde os tempos da ARPANet, quando interligava centros de pesquisa dos militares estadunidenses, para o que ela é atualmente? Essa reflexão pode não parecer necessária, mas de fato é essencial. A Internet só é tão útil, interessante e benéfica por conta de um determinado conjunto de características, de propriedades. Suas características podem mudar com o tempo, então é importante

entender quais propriedades são realmente importantes, e preservá-las! Você deve estar pensando “o que isso tem a ver com o IPv6?”... Em breve chegaremos lá, mas é importante antes destacar algumas características importantes da Internet.

Boa parte das características mais interessantes da rede mundial é resultado da forma como ela foi projetada para funcionar. Na Internet, cada computador tem um número que o identifica de forma única e sem possibilidade de confusão ou duplicação: o endereço IP. Ela foi projetada de forma que qualquer um dos computadores pode iniciar uma conversa, mandar informação para qualquer outro. Devemos lembrar que atualmente os “computadores” vêm em diversos modelos e formatos. Quando falamos em computadores, podemos na verdade nos referir tanto a *desktops* e *notebooks*, quanto a telefones celulares, carros, ou diversos outros tipos de equipamentos conectados. . . Essa propriedade da Internet que permite a qualquer dispositivo comunicar-se diretamente com qualquer outro é chamada de **conectividade fim a fim**.

A informação é sempre dividida em pequenos pedaços, os pacotes, e identificada com o endereço de origem e destino. Equipamentos na rede dos usuários e nas diversas outras redes que se unem e colaboram para formar a rede mundial estão encarregados de encontrar o caminho mais adequado e fazer cada um desses pacotes chegar ao destino correto. É importante notar que esses equipamentos que encaminham os pacotes e formam o núcleo da rede, os roteadores, realmente não fazem muita coisa. Isto não significa que sua função possa ser menosprezada, mas por simplicidade eles só têm uma função: enviar os pacotes pelo caminho correto até o destino! Isso quer dizer que o núcleo da rede não tem muita inteligência. Dessa forma fica mais fácil gerenciá-lo e fazê-lo crescer. . . De fato, isso deu muito certo. A Internet cresceu, nos últimos anos, de uma rede que interligava alguns poucos centros de pesquisa estadunidenses para a grande rede mundial que conhecemos e utilizamos hoje, que está em praticamente todo lugar. Essa propriedade da Internet é chamada de **simplicidade do núcleo da rede**.

Por causa da conectividade fim a fim e da simplicidade do núcleo da rede, que são propriedades diretamente derivadas da forma ela foi projetada para funcionar, os serviços interessantes da Internet, como páginas *Web*, aplicações, redes sociais, jogos, e tantos outros, funcionam nos servidores e computadores dos usuários, incluindo usuários domésticos, empresas de todos os tamanhos, governos, universidades, etc., nas extremidades da rede. Isso quer dizer que para a rede as informações são apenas pacotes e os pacotes são todos iguais entre si. Quer dizer também que qualquer um pode inventar novas aplicações, serviços ou protocolos para a rede: basta escrever um novo software e distribuí-lo. Não há necessidade de pedir permissão a quem controla a rede. Não existe tal controle centralizado. Essa é uma das razões da Internet ser tão propícia à **inovação**, aos **novos negócios** e aos **novos entrantes**. É uma das razões dela ser tão útil e interessante. A **conectividade fim a fim** e a **simplicidade do núcleo da rede** estão, portanto, entre as características da rede que são fundamentais e que devemos preservar.

É aqui que entra o IPv6. Desde 1983 a Internet é baseada no IP versão 4, ou IPv4. “IP” é abreviação de Protocolo Internet e um protocolo nada mais é do que um conjunto de regras que os computadores usam para conversar. Como visto anteriormente, uma das funções do Protocolo Internet é a de identificar cada dispositivo na rede mundial com um endereço numérico único, que chamamos de endereço IP. Toda a comunicação na rede depende desses endereços. Todos os serviços e aplicações usam o IP. É a principal tecnologia da Internet. Cada novo usuário ou, mais precisamente, cada novo computador, *tablet*, *smartphone*, *videogame*, *smart TV*, ou outro dispositivo na rede precisa de um novo endereço IP. Mas eles são finitos e praticamente já se esgotaram.

No Brasil e na América Latina já não temos mais IPs livres para conectar novos usuários. Por isso existe o IPv6. É uma nova versão do IP, com muito mais endereços, que foi projetada para substituir o IPv4 na rede. Só que o IPv6 é suficientemente diferente do IPv4 para que eles não sejam compatíveis, ou seja, eles não interoperam. Não dá para usar IPv4 numa parte da Internet e IPv6 em outra, porque esses diferentes pedaços, simplesmente, não poderiam se comunicar. Essa foi uma decisão de projeto necessária para acrescentar algumas características interessantes

no novo protocolo. Contudo, isso complica um pouco a transição. No entanto, há uma solução relativamente simples: por um tempo, vamos usá-los em paralelo. O caminho é usar ambos simultaneamente por alguns anos e, depois, ir desativando o protocolo antigo aos poucos. Muitos *sites* importantes, por exemplo, já funcionam tanto com o IPv6, como com o protocolo antigo.

O fim do IPv4 não aconteceu de uma hora para outra. No início da década de 1990, antes ainda da utilização da Internet “explodir” em todo o mundo, como um grande sucesso, já se sabia que isso aconteceria. Nessa mesma década, foi desenvolvido o IPv6 como solução. O IPv6 foi padronizado em 1998, com a proposta inicial de fazer uma transição gradual, antes ainda do esgotamento do IPv4. Tecnologias como NAT, CIDR e DHCP foram também desenvolvidas nessa época e implantadas com bastante sucesso na rede para dar mais tempo a essa transição. Contudo, a ideia de mudança gradual falhou. Tecnicamente era bastante simples e adequada, mas olhando agora em retrospectiva fica fácil perceber que poucos estariam dispostos a investir na adoção de uma tecnologia que resolveria um problema que iria acontecer de fato só dali a alguns anos. Que administrador de TI, por exemplo, no ano de 2001, pensaria em investir tempo e dinheiro para adotar o IPv6, sabendo que não teria vantagens relevantes se fizesse isso e só teria problemas, caso não fizesse nada, possivelmente dali a 10 ou 15 anos? A maior parte das redes que formam a Internet resolveu esperar até o último momento, até a transição realmente ser necessária.

Algo bastante complexo na transição para o IPv6 é justamente saber quando ela é realmente necessária. Não é possível marcar uma “data da virada” única. A Internet é grande demais, complexa demais, descentralizada demais, para isso. Até a data de publicação deste livro, muitas redes ainda adiam a adoção do protocolo, pensando que é um problema para o futuro. O fato é que o tempo ideal para implantarmos IPv6 já passou. Era bastante simples fazer isso enquanto ainda havia endereços IPv4 livres para suportar o crescimento da Internet. Agora não há mais. Criamos um enorme complicador, do ponto de vista técnico, que coloca em risco a própria rede, ameaçando algumas das características que deveríamos preservar.

Na falta de endereços IPv4 para conectar novos usuários, os provedores de acesso à Internet trilharão dois caminhos em paralelo. Eles conectarão novos usuários com IPv6. Eles também usarão o protocolo antigo, o IPv4, mas como não terão mais endereços livres, vão compartilhá-los entre diferentes usuários. A tecnologia que permite que dezenas de dispositivos dividam o mesmo endereço IPv4 é o CGNAT, ou *Carrier Grade NAT*. É uma variação da tecnologia de compartilhamento que hoje é comum nas nossas casas e empresas, mas o fato de ser usada na rede do provedor **quebra as propriedades de conectividade fim a fim e de simplicidade no núcleo da rede**. O CGNAT é um mal necessário, mas se ele vier desacompanhado do IPv6 pode ser um verdadeiro desastre para a Internet.

O CGNAT aumenta o custo da rede, pois acrescenta equipamentos caros e complexos, que podem ser pontos de falha importantes, em seu núcleo. Há aplicações que não funcionam bem com a tecnologia, como conferências de voz e vídeo, compartilhamento *peer to peer* e alguns jogos. Com o CGNAT um mesmo IP pode identificar dezenas ou centenas de dispositivos simultaneamente, o que dificulta consultas à bases de geolocalização, bases de reputação, e mesmo investigações criminais.

Todos os grandes provedores não terão outra escolha, senão usar o CGNAT ou tecnologia similar durante algum tempo. Alguns já começaram a usar o IPv6 ou começarão antes disso, o que é excelente. Outros, só planejam trabalhar com IPv6 algum tempo depois, o que é motivo de grande preocupação. Uma Internet baseada só no compartilhamento dos endereços antigos é algo que ninguém quer. No pior caso, poderíamos ter uma rede muito pior do que a que existe hoje: fragmentada, pouco propícia à inovação, aos novos entrantes e novos negócios.

O IPv6 é fundamental, então, para a expansão da Internet, possibilitando a continuidade da adição de novos usuários e o desenvolvimento da Internet das Coisas, interligando os mais diversos tipos de objetos inteligentes. Não é exagero dizer que o IPv6 é fundamental para a própria sobrevivência da Internet nos moldes em que a conhecemos atualmente.

as será que o IPv6 é realmente importante agora para a empresa, universidade ou outra entidade na qual o leitor está inserido? É importante para a rede que o leitor ajuda a administrar? A resposta é sim. Se o leitor atua em um provedor de acesso à Internet, ou gerencia um outro tipo de rede que não tem endereços IPv4 suficientes para continuar se expandindo, essa resposta é bastante óbvia. Mas, mesmo que uma determinada rede já esteja conectada à Internet e tenha IPs versão 4 suficientes para suportar sua expansão, nos próximos anos, ela não pode ficar à parte na transição para o IPv6. Há duas razões principais para isso.

A primeira razão é de que a rede não está isolada. A única escolha possível para muitas outras redes na Internet se expandirem é a implantação do IPv6. Se uma empresa, universidade, ou outra entidade tem serviços expostos na Internet, com uma página na *Web*, ou um servidor de *e-mails* próprio, é importante que estes funcionem também usando IPv6 o mais rapidamente possível. Assim, a conectividade com essas novas redes e novos usuários que terão o IPv6 como opção preferencial não será prejudicada. É por isso que os principais portais, buscadores e outros serviços na Internet já implantaram IPv6.

A segunda razão é que, de fato, há uma alta probabilidade de que a rede, qualquer rede, já trafegue pacotes IPv6. Se os usuários dessa rede têm versões recentes de Windows, Mac OS ou Linux em seus dispositivos, por exemplo, é importante notar que nestes sistemas operacionais o IPv6 está ativo por padrão. Há algum risco em usar o protocolo dessa forma, sem a devida preparação. Por exemplo, alguns computadores podem tentar, automaticamente e sem intervenção ou conhecimento do usuário ou administrador de redes, utilizar técnicas de tunelamento para obter conectividade IPv6 na Internet, contornando dispositivos de proteção e filtros. Não é possível simplesmente ignorar o IPv6.

Se o leitor ainda não estava convencido da necessidade do IPv6, e de que agora é a hora certa para implantá-lo, esperamos que tenha encontrado neste capítulo alguns bons argumentos para se convencer. Se já estava convencido, esperamos que esteja agora com essa convicção reforçada! Os demais capítulos deste livro são bem diferentes. Eles buscam trazer uma visão prática do funcionamento e do uso do IPv6 no dia a dia, por meio de uma série de experimentos. Este não é um livro apenas para

ser lido: deve-se praticar, realizando as experiências no emulador CORE. Estas experiências provavelmente não responderão a todas as dúvidas sobre o IPv6, mas certamente darão uma base bastante sólida, deixando o leitor confortável para começar a trabalhar com ele e implantá-lo nas redes que ajuda a gerenciar.

Mãos à obra!

Capítulo 1

Funcionalidades básicas

Experiência 1.1. *Neighbor Discovery Protocol: Neighbor Solicitation e Neighbor Advertisement*

Objetivo

Esta experiência tem como objetivo apresentar o funcionamento do mecanismo de descoberta de vizinhos do IPv6, que é responsabilidade do protocolo *Neighbor Discovery*. Isto será feito forçando a comunicação entre dois nós diretamente ligados entre si com um ping IPv6, utilizando o binário ping6. Será possível observar como as mensagens *Neighbor Solicitation* e *Neighbor Advertisement* são utilizadas para mapear o endereço físico e o endereço IP.

Para o presente exercício será utilizada a topologia descrita no arquivo: **1-01-NSNA.imn**.

Introdução teórica

A descoberta de vizinhança por meio do protocolo *Neighbor Discovery* no IPv6 é um procedimento realizado pelos nós de uma rede para descobrir endereços físicos dos dispositivos vizinhos presentes no mesmo enlace. A função deste protocolo se assemelha à função do ARP e do RARP no IPv4.

O procedimento é iniciado quando um dispositivo tenta enviar um pacote cujo endereço físico de destino é desconhecido. O nó solicitante envia uma mensagem *Neighbor Solicitation* (NS) para todos os nós do enlace pertencentes ao grupo *multicast solicited-node* (ff02::1:ffXX:XXXX), de modo que XX:XXXX são os últimos 24 *bits* do endereço IPv6 em que está interessado.

É possível notar que, por uma coincidência dos últimos 24 *bits*, é bastante provável que apenas o nó de destino faça realmente parte deste grupo. Isto é um *truque* interessante do IPv6 para diminuir o tráfego deste tipo de pacote na rede.

Na mensagem NS, o endereço IPv6 a ser resolvido é informado no campo *Target*. O campo *Source link-layer address* informa ao nó de destino o endereço MAC do nó de origem, poupando-o de ter que fazer o mesmo procedimento no sentido inverso.

O nó de destino, dono do IPv6 requisitado, ao receber este pacote, envia uma mensagem *Neighbor Advertisement* (NA) como resposta diretamente ao nó requisitante. O seu endereço físico será informado no campo *Target link-layer address*.

A informação de mapeamento entre endereços IP e endereços físicos é armazenada em uma tabela chamada *neighbor cache*. Nela também fica registrado o *status* de cada destino, informando se o mesmo é alcançável ou não.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-01-NSNA.imn** localizado no diretório `lab`, dentro do Desktop. A topologia de rede, representada pela Figura 1.1, deve aparecer.

O objetivo desta topologia de rede é representar o mínimo necessário para que a troca das mensagens NS e NA seja verificada.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação e verifique a configuração de endereços IPv6 nos nós `n1HostA` e `n2HostB`.

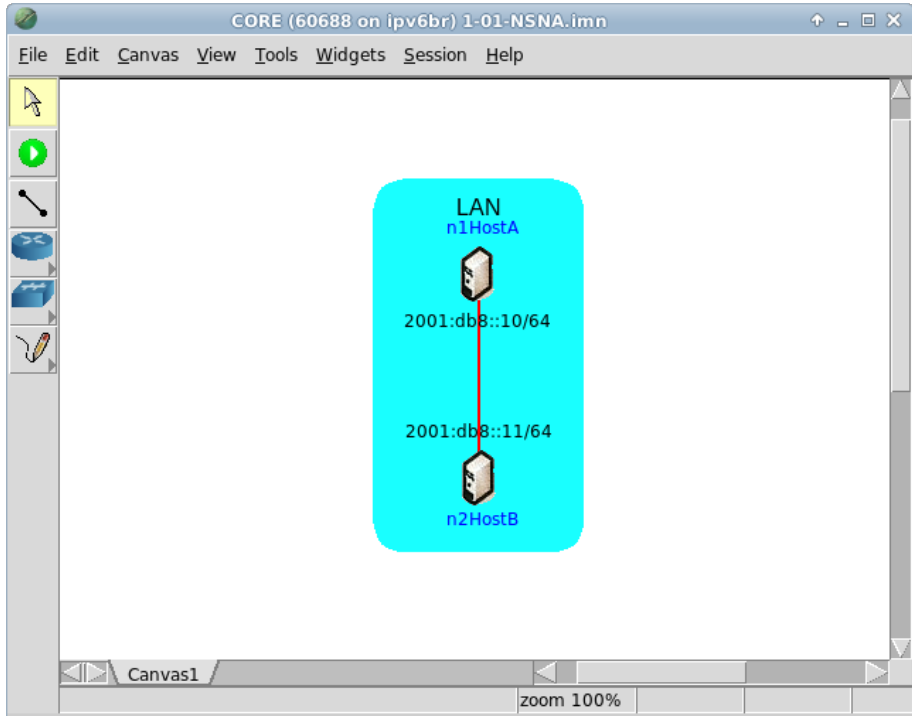


Figura 1.1: *topologia da Experiência 1.1 no CORE.*

3. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface `eth0` de `n1HostA`. As instruções de coleta de pacotes utilizando `tcpdump` ou Wireshark se encontram no Apêndice C.
 - (b) Um teste de conectividade IPv6 entre `n1HostA` e `n2HostB` com um `ping6`.
4. Efetue a análise dos pacotes coletados. Aplique o filtro `icmpv6` no Wireshark e procure pelos pacotes NS e NA. Analise-os e veja se os dados contidos nos pacotes conferem com o que foi passado na teoria.

Campos importantes do NS, representado na Figura 1.2:

Destination (camada Ethernet)

O destino é o endereço (`33:33:ff:00:00:11`), sendo que o prefixo `33:33` indica que a mensagem é um *multicast* na camada Ethernet. O sufixo `ff:00:00:11` indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

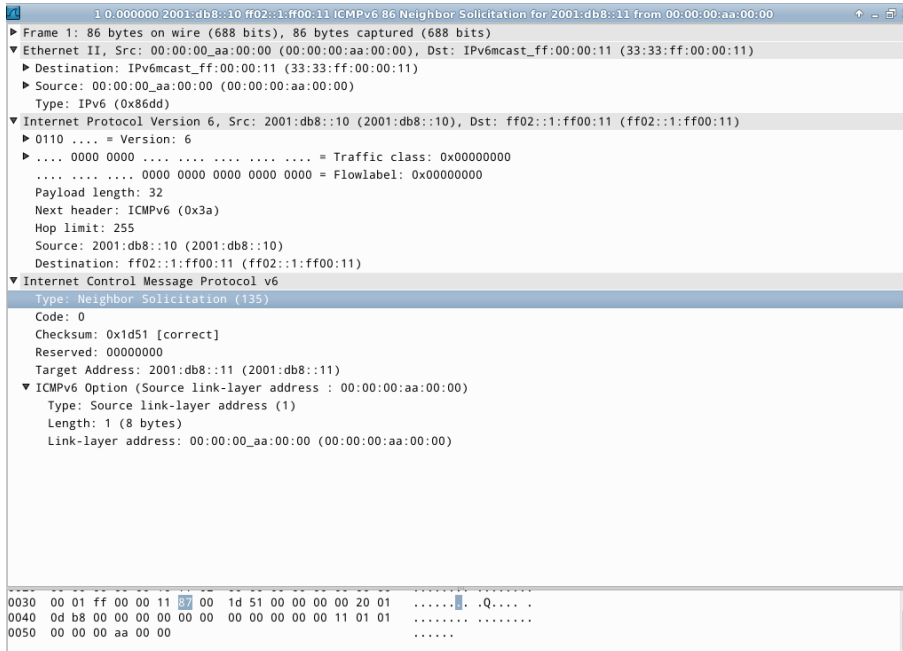


Figura 1.2: pacote NS mostrado no Wireshark.

Source (camada Ethernet)

A origem é o endereço MAC da interface do dispositivo que enviou a solicitação (00:00:00:aa:00:00).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se a uma mensagem ICMPv6.

Source (camada IPv6)

A origem é o endereço IP da interface diretamente ligada ao enlace em que se faz a requisição (2001:db8::10).

Destination (camada IPv6)

O destino é o endereço *multicast solicited-node* (ff02::1:ff00:11).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 135 (*Neighbor Solicitation*).

ICMPv6 option (camada ICMPv6)

Indica as opções do pacote ICMPv6:

- *Source link-layer address*

Type

Indica o tipo de opção. Neste caso, *Source link-layer address*.

Link-layer address

Indica o endereço MAC da interface de origem da mensagem.

Campos importantes do pacote NA, representado na Figura 1.3:

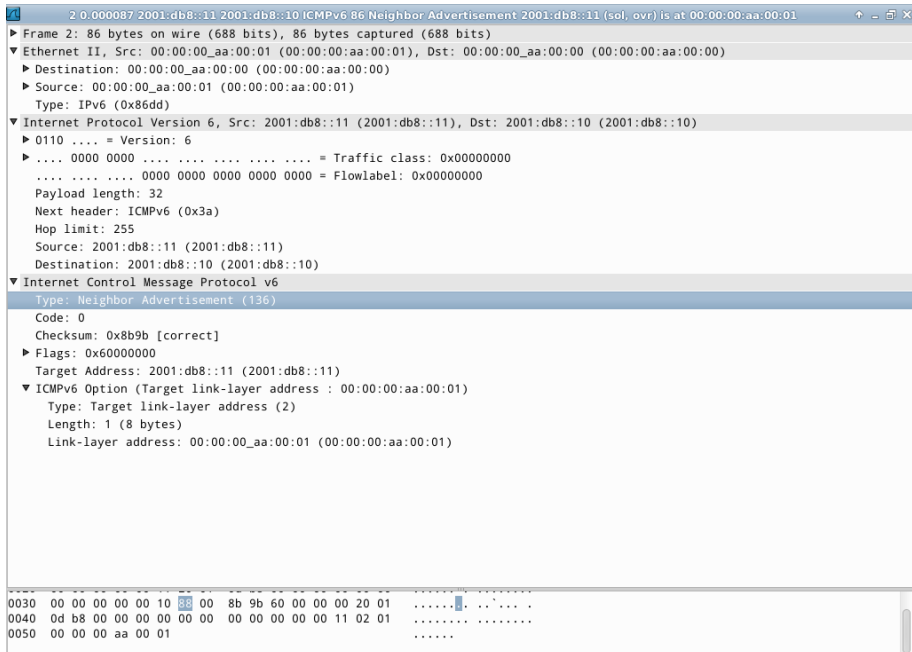


Figura 1.3: pacote NA mostrado no Wireshark.

Destination (camada Ethernet)

O endereço MAC do nó requisitante que foi obtido por meio da mensagem NS enviada anteriormente (00:00:00:aa:00:00).

Source (camada Ethernet)

A origem é o endereço MAC da interface do dispositivo n1HostA que enviou a resposta (00:00:00:aa:00:01).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se a uma mensagem ICMPv6.

Source (camada IPv6)

A origem é o endereço IP da interface diretamente ligada ao enlace em que a requisição foi recebida (2001:db8::11).

Destination (camada IPv6)

Diferentemente da mensagem NS, a mensagem NA possui como destino o endereço IPv6 global do nó requisitante (2001:db8::10).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 136 (*Neighbor Advertisement*).

Flags (camada ICMPv6)

Uma mensagem NA possui três *flags*:

- Indica se quem está enviando é um roteador. Neste caso, o valor marcado é 0, pois não é um roteador.
- Indica se a mensagem é uma resposta a um NS. Neste caso, o valor marcado é 1, pois é uma resposta.
- Indica se a informação carregada na mensagem é uma atualização de endereço de algum nó da rede. Neste caso, o valor marcado é 1, pois está informando o endereço pela primeira vez.

Target Address (camada ICMPv6)

Indica o endereço IP associado às informações das *flags*. Neste caso, é o próprio endereço da interface do dispositivo n1HostA (2001:db8::11).

ICMPv6 option (camada ICMPv6)

Indica as opções do pacote ICMPv6:

- *Target link-layer address*

Type

Indica o tipo de opção. Neste caso, *Target link-layer address*.

Link-layer address

Indica o endereço MAC da interface do dispositivo n1HostA (00:00:00:aa:00:01).

5. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 1.2. *Neighbor Discovery Protocol: Router Solicitation*

Objetivo

Esta experiência possui como objetivo apresentar o funcionamento do mecanismo de descoberta de roteadores. Para isto, ela foi dividida em duas partes.

A primeira parte mostra como um nó envia uma mensagem *Router Solicitation* (RS) quando sua interface de rede é ativada, tendo como resposta do roteador uma mensagem *Router Advertisement* (RA).

O roteiro da Experiência 1.3 mostra o anúncio periódico do roteador para a rede com mensagens RA.

O presente exercício utiliza a topologia descrita no arquivo: **1-02-RS.imn**.

Introdução teórica

A descoberta de roteadores é um procedimento realizado pelos nós no momento em que se conectam ou se reconectam a uma rede, para descobrir características do enlace e rotas de comunicação.

O mecanismo é iniciado com o envio da mensagem RS direcionada a todos os roteadores no enlace, por meio do endereço de grupo *multicast all-routers* (ff02::2).

A mensagem também informa o endereço físico do nó que a envia, poupando o roteador de fazer o procedimento de descoberta de vizinhança antes de enviar uma resposta.

Ao receber a solicitação, o roteador responde com uma mensagem RA. O endereço de origem é seu *link-local*.

O roteador pode enviar mensagens RA também sem receber qualquer solicitação, de forma periódica, dependendo de sua configuração. Neste caso, ela é enviada para o grupo *multicast all-nodes*.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-02-RS.imn**, localizado no diretório `lab`, dentro do Desktop. A topologia de rede, representada pela Figura 1.4, deve aparecer.

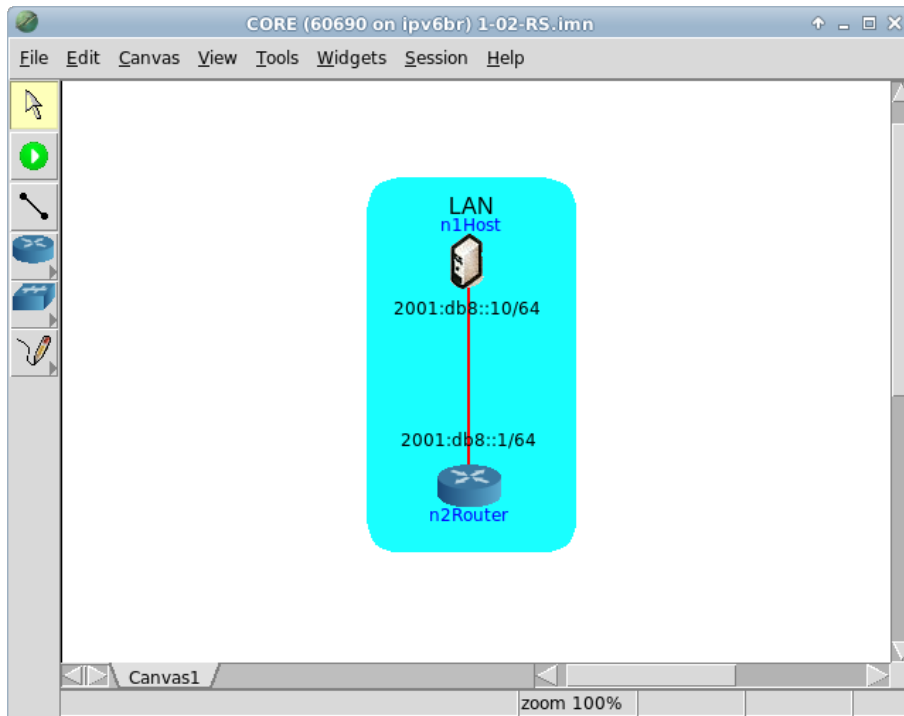


Figura 1.4: topologia da *Experiência 1.2* no CORE.

O objetivo desta topologia de rede é representar o mínimo necessário para que a mensagem RS seja verificada.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós `n1Host` e `n2Router` e a conectividade entre eles.
3. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface `eth0` de `n2Router`. As instruções de coleta de pacotes utilizando `tcpdump` ou `Wireshark` se encontram no Apêndice C.

- (b) Abra um terminal de n1Host com um duplo-clique e desabilite a interface eth0 temporariamente por meio dos comandos:

```
# ip link set eth0 down
# ip link set eth0 up
```

Estes comandos desabilitam e, em seguida, habilitam a interface eth0 de n1Host e atribuem o mesmo endereço IPv6 utilizado previamente. Isto é realizado para forçar o envio da mensagem RS por n1Host. O resultado dos comandos é representado pela Figura 1.5.



```
n1Host
root@n1Host:/tmp/pycore.36788/n1Host.conf# ip link set eth0 down
root@n1Host:/tmp/pycore.36788/n1Host.conf# ip link set eth0 up
root@n1Host:/tmp/pycore.36788/n1Host.conf# █
```

Figura 1.5: resultado da desabilitação temporária da interface eth0 de n1Host.

4. As mensagens RA podem informar diversos parâmetros da rede para que os nós se configurem automaticamente. Nesta experiência, nenhum parâmetro foi configurado nas mensagens RA enviadas pelo n2Router. No entanto, por padrão, o roteador se anuncia como *gateway* da rede por meio destas mensagens. Isso pode ser verificado executando o seguinte passo:

Abra um terminal de n1Host com um duplo-clique e utilize o seguinte comando para verificar sua tabela de rotas:

```
# ip -6 route show
```

O resultado dos comandos é representado pela Figura 1.6.



```
n1Host
root@n1Host:/tmp/pycore.46757/n1Host.conf# ip -6 route show
fe80::/64 dev eth0 proto kernel metric 256
default via fe80::200:ff:feaa:1 dev eth0 proto kernel metric 1024 expires 11sec
root@n1Host:/tmp/pycore.46757/n1Host.conf# █
```

Figura 1.6: verificação das rotas de n1Host.

5. Efetue a análise dos pacotes capturados. Aplique o filtro icmpv6 no Wireshark e procure pelos pacotes RS e RA. Analise-os e veja se os dados contidos nos pacotes conferem com o que foi passado na teoria.

Campos importantes do pacote RS, representado pela Figura 1.7:

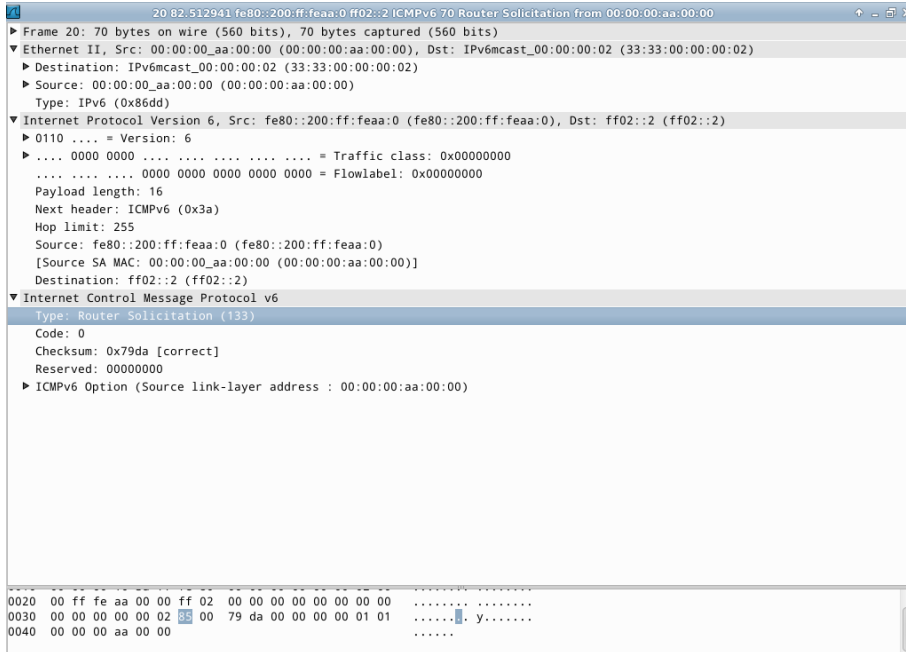


Figura 1.7: pacote RS mostrado no Wireshark.

Destination (camada Ethernet)

O destino é o endereço (33:33:00:00:00:02), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:00:00:02 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

Indica o endereço MAC da interface do dispositivo que enviou a solicitação (00:00:00:aa:00:00).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se a uma mensagem ICMPv6.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface que fez a requisição (fe80::200:ff:feaa:0).

Destination (camada IPv6)

O destino é o endereço *multicast all-routers* (ff02::2).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 133 (*Router Solicitation*).

ICMPv6 option (camada ICMPv6)

Indica as opções do pacote ICMPv6:

- *Source link-layer address*

Type

Indica o tipo de opção. Neste caso, *Source link-layer address*.

Link-layer address

Indica o endereço MAC da interface de origem da mensagem.

Campos importantes do pacote RA, representado pela Figura 1.8:

Destination (camada Ethernet)

O destino é o endereço (33:33:00:00:00:01), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:00:00:01 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

Indica endereço MAC da interface do roteador que originou a resposta (00:00:00:aa:00:01).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

```

21.82.513586 fe80::200:ff:feaa:1 ff02::1 ICMPv6 78 Router Advertisement from 00:00:00:aa:00:01
  Frame 21: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
  Ethernet II, Src: 00:00:00_aa:00:01 (00:00:00:aa:00:01), Dst: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
  Internet Protocol Version 6, Src: fe80::200:ff:feaa:1 (fe80::200:ff:feaa:1), Dst: ff02::1 (ff02::1)
    0110 .... = Version: 6
    .... 0000 0000 .... = Traffic class: 0x00000000
    .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 24
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source: fe80::200:ff:feaa:1 (fe80::200:ff:feaa:1)
    [Source SA MAC: 00:00:00_aa:00:01 (00:00:00:aa:00:01)]
    Destination: ff02::1 (ff02::1)
  Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x38c2 [correct]
    Cur hop limit: 64
    Flags: 0x00
      0... .... = Managed address configuration: Not set
      .0... .... = Other configuration: Not set
      ..0... .... = Home Agent: Not set
      ...0 0... = Prf (Default Router Preference): Medium (0)
      .... .0.. = Proxy: Not set
      .....0.. = Reserved: 0
    Router lifetime (s): 15
    Reachable time (ms): 0
    Retrans timer (ms): 0
    ICMPv6 Option (Source link-layer address : 00:00:00:aa:00:01)
      Type: Source link-layer address (1)
      Length: 1 (8 bytes)
      Link-layer address: 00:00:00_aa:00:01 (00:00:00:aa:00:01)
  0020 00 ff fe aa 00 01 ff 02 00 00 00 00 00 00 00 00 .....
  0030 00 00 00 00 00 01 38 c2 40 00 00 0f 00 00 ..... 8.@.....
  0040 00 00 00 00 00 00 01 01 00 00 00 aa 00 01 .....
  
```

Figura 1.8: pacote RA mostrado no Wireshark.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se à uma mensagem ICMPv6.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface que enviou a resposta, sendo neste caso o roteador (fe80::200:ff:feaa:1).

Destination (camada IPv6)

O destino é o endereço *multicast all-nodes* (ff02::1).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 134 (*Router Advertisement*).

ICMPv6 option (camada ICMPv6)

Indica as opções do pacote ICMPv6:

- Source link-layer address

Type

Indica o tipo de opção. Neste caso, *Source link-layer address*.

Link-layer address

Indica o endereço MAC da interface de origem da mensagem.

6. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 1.3. *Neighbor Discovery Protocol: Router Advertisement*

Objetivo

Esta experiência possui como objetivo apresentar o funcionamento do mecanismo de descoberta de roteadores e foi dividida em duas partes.

O roteiro da Experiência 1.2 mostra como um nó envia uma mensagem RS quando sua interface de rede é ativada, tendo como resposta do roteador um RA.

Esta segunda parte mostra o anúncio periódico do roteador para a rede com mensagens RA.

O presente exercício utiliza a topologia descrita no arquivo: **1-03-RA.imn**.

Introdução teórica

Veja a introdução teórica da Experiência 1.2.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-03-RA.imn** localizado no diretório `lab`, dentro do Desktop. A topologia de rede, representada pela Figura 1.9, deve aparecer.

O objetivo desta topologia de rede é representar o mínimo necessário para que a mensagem RA seja verificada.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós `n1Host` e `n2Router` e a conectividade entre eles.

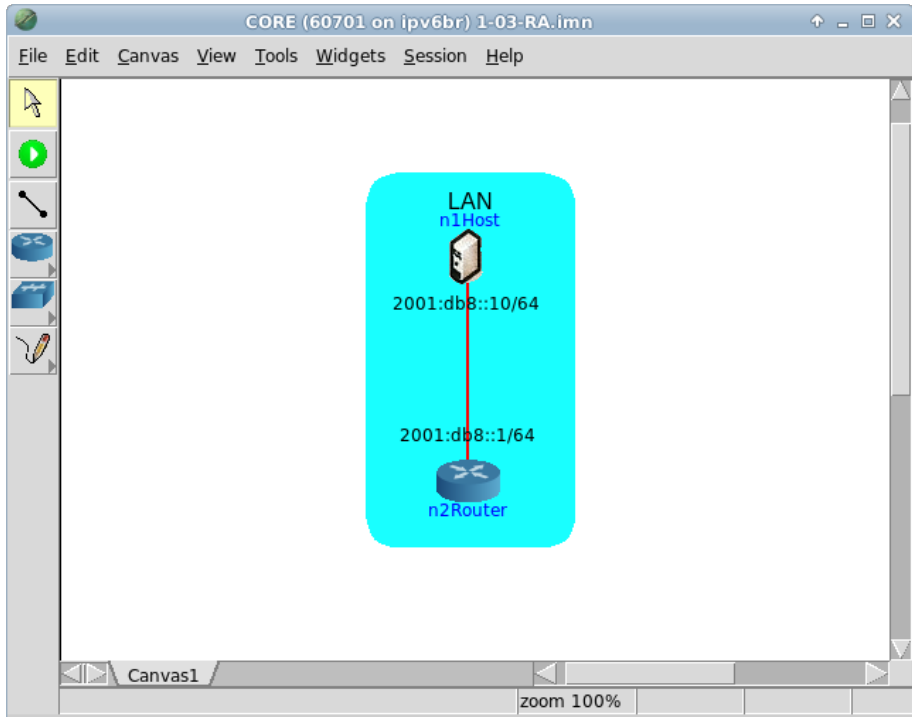


Figura 1.9: *topologia da Experiência 1.3 no CORE.*

3. Configure o roteador de modo que o Quagga envie a mensagem RA.
 - (a) Abra um terminal de n2Router com um duplo-clique e verifique o conteúdo do arquivo de configuração do Quagga. Utilize o seguinte comando:

```
# cat /usr/local/etc/quagga/Quagga.conf
```

O resultado do comando é representado pela Figura 1.10.

```
root@n2Router:/tmp/pycore.34135/n2Router.conf# cat /usr/local/etc/quagga/Quagga.conf
interface eth0
!
root@n2Router:/tmp/pycore.34135/n2Router.conf#
```

Figura 1.10: *verificação do arquivo de configuração do Quagga antes de sua edição.*

- (b) Ainda no terminal de n2Router, edite o arquivo de configuração do Quagga, localizado em /usr/local/etc/quagga/Quagga.conf, de modo a adicionar as linhas em **negrito**.

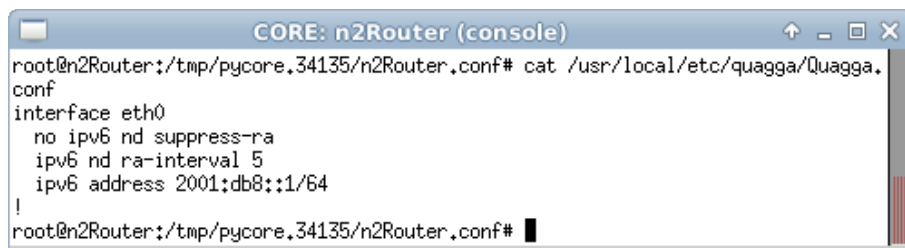
```
interface eth0
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 5
  ipv6 address 2001:db8::1/64
!
```

As três linhas devem ser inseridas dentro do escopo da interface, isto é, entre as linhas `interface eth0` e `!`. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

- (c) Ainda no terminal de `n2Router`, verifique novamente o conteúdo do arquivo de configuração do Quagga, com o comando:

```
# cat /usr/local/etc/quagga/Quagga.conf
```

O resultado do comando é representado pela Figura 1.11.



```
root@n2Router:~/tmp/pycore.34135/n2Router.conf# cat /usr/local/etc/quagga/Quagga.conf
interface eth0
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 5
  ipv6 address 2001:db8::1/64
!
root@n2Router:~/tmp/pycore.34135/n2Router.conf#
```

Figura 1.11: *verificação do arquivo de configuração do Quagga após sua edição.*

4. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface `eth0` de `n1Host`. As instruções de coleta de pacotes utilizando `tcpdump` ou `Wireshark` se encontram no Apêndice C.
 - (b) Abra um terminal de `n2Router` com um duplo-clique e utilize o seguinte comando para iniciar o Quagga com as novas configurações:

```
# ./boot.sh
```

Este comando reinicializa os serviços associados ao roteador, incluindo o serviço de roteamento Quagga. Isto é realizado para que o roteador envie a mensagem RA. O resultado do comando é representado pela Figura 1.12.

```

CORE: n2Router (console)
root@n2Router:/tmp/pycore.34135/n2Router.conf# ./boot.sh
net.ipv4.conf.all.forwarding = 1
net.ipv6.conf.all.forwarding = 1
net.ipv4.conf.all.send_redirects = 0
root@n2Router:/tmp/pycore.34135/n2Router.conf#

```

Figura 1.12: resultado esperado da inicialização do Quagga com a nova configuração.

5. Efetue a análise dos pacotes capturados. Aplique o filtro icmpv6 no Wireshark e procure pelos pacotes RA.

Analise os pacotes RA e veja se os dados contidos nos pacotes conferem com a teoria.

Campos importantes do pacote RA, representado pela Figura 1.13:

```

1.0.000000 fe80::200:ff:feaa:1 ff02::1 ICMPv6 78 Router Advertisement from 00:00:00:aa:00:01
  Frame 1: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
  Ethernet II, Src: 00:00:00:aa:00:01 (00:00:00:aa:00:01), Dst: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
    Destination: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
    Source: 00:00:00:aa:00:01 (00:00:00:aa:00:01)
    Type: IPv6 (0x86dd)
  Internet Protocol Version 6, Src: fe80::200:ff:feaa:1 (fe80::200:ff:feaa:1), Dst: ff02::1 (ff02::1)
    0110 .... = Version: 6
    .... 0000 0000 .... = Traffic class: 0x00000000
    .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 24
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source: fe80::200:ff:feaa:1 (fe80::200:ff:feaa:1)
    [Source SA MAC: 00:00:00:aa:00:01 (00:00:00:aa:00:01)]
    Destination: ff02::1 (ff02::1)
  Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x31c9 [correct]
    Cur hop limit: 64
    Flags: 0x00
    Router lifetime (s): 1800
    Reachable time (ms): 0
    Retrans timer (ms): 0
    ICMPv6 Option (Source link-layer address : 00:00:00:aa:00:01)
      Type: Source link-layer address (1)
      Length: 1 (8 bytes)
      Link-layer address: 00:00:00:aa:00:01 (00:00:00:aa:00:01)

```

Figura 1.13: pacote RA mostrado no Wireshark.

Destination (camada Ethernet)

O destino é o endereço (33:33:00:00:00:01), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:00:00:01 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do roteador que enviou a resposta (00:00:00:aa:00:01).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se à uma mensagem ICMPv6.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface que enviou a resposta, sendo neste caso o roteador (fe80::200:ff:feaa:1).

Destination (camada IPv6)

O destino é o endereço *multicast all-nodes* (ff02::1).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 134 (*Router Advertisement*).

ICMPv6 option (camada ICMPv6)

Indica as opções do pacote ICMPv6:

- *Source link-layer address*

Type

Indica o tipo de opção. Neste caso, *Source link-layer address*.

Link-layer address

Indica o endereço MAC da interface a partir da qual a mensagem de *Router Advertisement* foi enviada, sendo neste caso 00:00:00:aa:00:01.

6. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 1.4. *Neighbor Discovery Protocol*: detecção de endereços duplicados

Objetivo

Esta experiência apresenta o funcionamento do mecanismo de detecção de endereços duplicados. Para isto, troca-se o endereço de um dos nós da topologia para um endereço em uso por outro nó. É possível observar a detecção do endereço duplicado pela troca de mensagens NS e NA. Observa-se também que o funcionamento do nó cujo endereço foi duplicado não é afetado.

Para a realização do presente exercício será utilizada a topologia descrita no arquivo: **1-04-DAD.imn**.

Introdução teórica

A detecção de endereços duplicados (*Duplicate Address Detection* – DAD) é um procedimento realizado sempre que um novo endereço é atribuído a uma interface. Isto vale tanto para endereços atribuídos manualmente, quanto por autoconfiguração. Tanto na hora em que o dispositivo é ligado, como quando já está em funcionamento e um novo endereço é adicionado.

O mecanismo é parecido com o da descoberta de vizinhança, com a diferença de que o nó tenta descobrir o endereço físico de seu próprio endereço IP, antes de efetivamente usá-lo. Se alguém responder, é porque o endereço já está em uso por outro nó.

Isso é feito enviando uma mensagem NS, como na descoberta de vizinhança. Mas o endereço de origem é :: (não especificado).

Caso receba uma mensagem NA, o nó interrompe o processo de configuração e o endereço não poderá ser utilizado. Nessa situação, o conflito só é solucionado manualmente, com adição de um novo endereço.

Caso um determinado tempo de espera seja ultrapassado e não seja recebida nenhuma mensagem, o dispositivo poderá então finalizar sua configuração. O valor padrão para tal espera é de um segundo.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-04-DAD.imn** localizado no diretório `lab`, dentro do Desktop. A topologia de rede, representada pela Figura 1.14, deve aparecer.

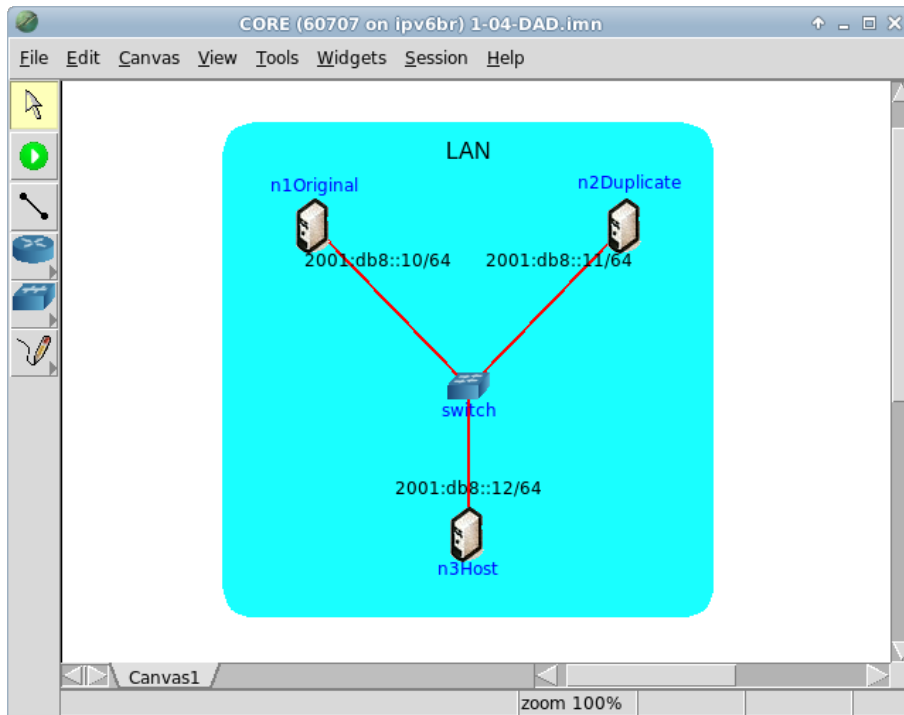


Figura 1.14: topologia da *Experiência 1.4* no CORE.

O objetivo dessa topologia de rede é representar o mínimo necessário para que a troca das mensagens relativas à duplicidade de endereços seja verificada.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós `n1Original`, `n2Duplicate` e `n3Host` e a conectividade entre eles.
3. Em paralelo, efetue:

- (a) A coleta dos pacotes trafegados na interface eth0 de n1original. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
- (b) Abra um terminal de n2Duplicate com um duplo-clique e utilize os seguintes comandos:

```
# ip addr del 2001:db8::11/64 dev eth0
# ip addr add 2001:db8::10/64 dev eth0
```

O resultado dos comandos é representado pela Figura 1.15.



```
CORE: n2Duplicate (console)
root@n2Duplicate:/tmp/pycore.34135/n2Duplicate.conf# ip addr del 2001:db8::11/64
dev eth0
root@n2Duplicate:/tmp/pycore.34135/n2Duplicate.conf# ip addr add 2001:db8::10/64
dev eth0
root@n2Duplicate:/tmp/pycore.34135/n2Duplicate.conf# █
```

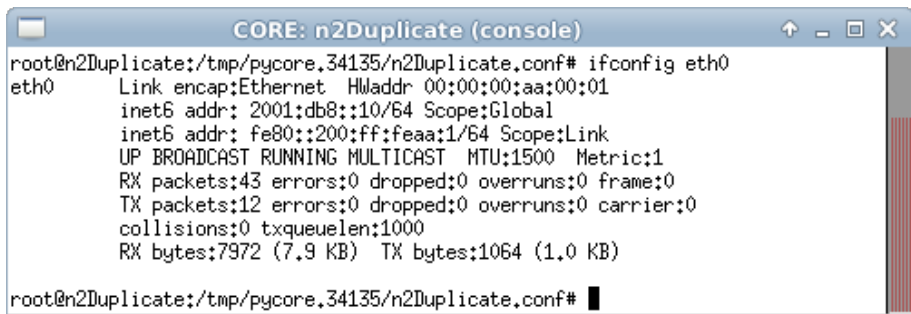
Figura 1.15: resultado esperado da atribuição de um endereço duplicado.

Estes comandos removem o endereço IPv6 anterior e tentam atribuir um endereço duplicado.

4. Verifique o resultado da atribuição do endereço duplicado em n2Duplicate.
- (a) Abra um terminal de n2Duplicate com um duplo-clique e verifique a atribuição do endereço duplicado. Utilize o seguinte comando:

```
# ifconfig eth0
```

O resultado do comando é representado pela Figura 1.16.



```
CORE: n2Duplicate (console)
root@n2Duplicate:/tmp/pycore.34135/n2Duplicate.conf# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:00:00:aa:00:01
          inet6 addr: 2001:db8::10/64 Scope:Global
          inet6 addr: fe80::200:ff:feaa:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:43 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7972 (7.9 KB)  TX bytes:1064 (1.0 KB)

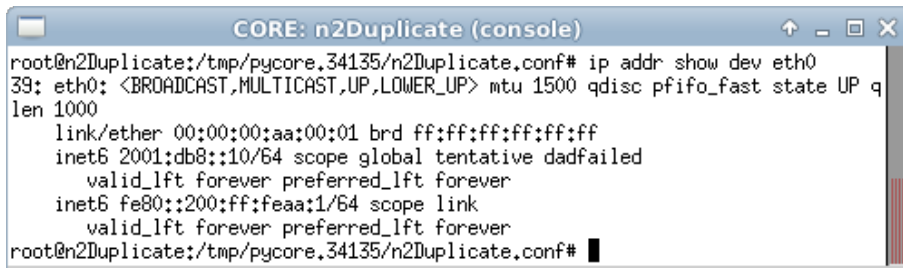
root@n2Duplicate:/tmp/pycore.34135/n2Duplicate.conf# █
```

Figura 1.16: verificação dos endereços atribuídos a n2Duplicate utilizando o comando ifconfig.

- (b) Ainda no terminal de n2Duplicate, verifique a atribuição do endereço duplicado, com o comando:

```
# ip addr show dev eth0
```

O resultado do comando é representado pela Figura 1.17.



```
root@n2Duplicate:~/tmp/pycore.34135/n2Duplicate.conf# ip addr show dev eth0
39: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP q
len 1000
    link/ether 00:00:00:aa:00:01 brd ff:ff:ff:ff:ff:ff
    inet6 2001:db8::10/64 scope global tentative dadfailed
        valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feaa:1/64 scope link
        valid_lft forever preferred_lft forever
root@n2Duplicate:~/tmp/pycore.34135/n2Duplicate.conf# █
```

Figura 1.17: *verificação dos endereços atribuídos a n2Duplicate utilizando o comando ip.*

A detecção de endereço duplicado é um procedimento definido no IPv6 e o objetivo de evitar o uso de um mesmo endereço por dois dispositivos distintos.

Observe as linhas referentes ao endereço 2001:db8::10 na saída dos comandos `ifconfig` e `ip` utilizados. Note a informação *tentative dadfailed* contida na saída do comando `ip`. Esta informação indica que foi detectada a tentativa de uso de um endereço duplicado e, pelo fato da duplicidade de endereço, o mesmo não foi atribuído à interface `eth0` de `n2Duplicate`.

Já a saída do comando `ifconfig` não indica tal falha. Este é um dos motivos para a depreciação do comando `ifconfig` e a recomendação do comando `ip` para a mesma finalidade de verificação de informações relativas às interfaces de rede quando se utiliza o sistema operacional Linux.

O procedimento da detecção de endereço duplicado também é abordado na Experiência 3.1.

5. Efetue a análise dos pacotes coletados.
- (a) Primeiro, analise os pacotes coletados da interface `eth0` de `n10original` durante o passo 3.

Aplique o filtro `icmpv6` no Wireshark e procure pelos pacotes NS, cujo *Source* é `:::`; NA em resposta ao NS anterior; e *echo reply*. Analise-os e veja se os dados contidos nos pacotes conferem com o que foi passado na teoria.

Campos importantes do pacote NS, representado pela Figura 1.18:

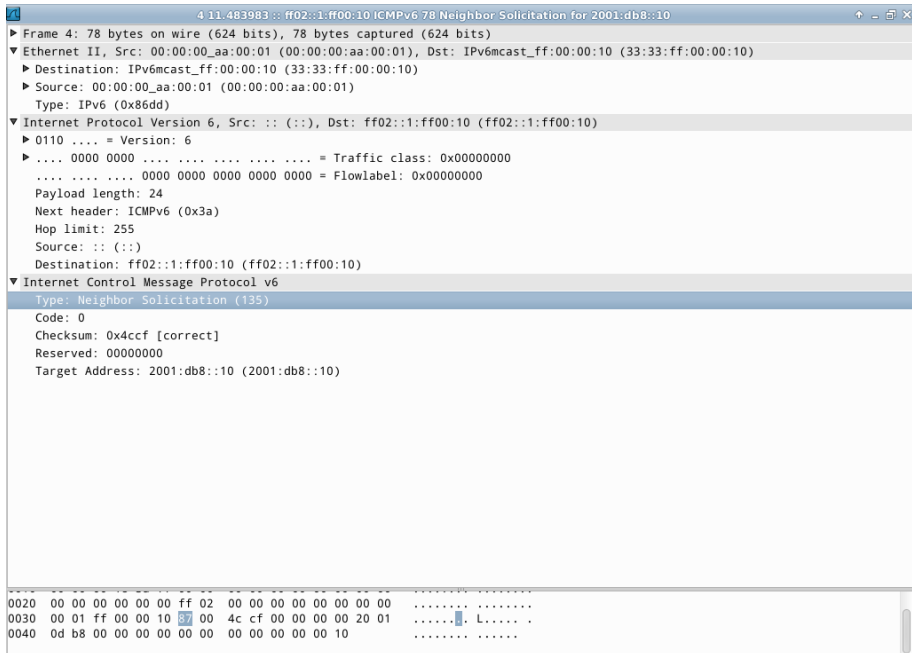


Figura 1.18: pacote NS capturado em `n1original`, mostrado no Wireshark.

Destination (camada Ethernet)

O destino é o endereço `(33:33:ff:00:00:10)`, sendo que o prefixo `33:33` indica que a mensagem é um *multicast* na camada Ethernet. O sufixo `ff:00:00:10` indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do dispositivo que enviou a solicitação `(00:00:00:aa:00:01)`.

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se a uma mensagem ICMPv6.

Source (camada IPv6)

A origem não é especificada, sendo utilizado o endereço (::).

Destination (camada IPv6)

O destino é o endereço *multicast solicited-node* (ff02::1:ff00:10).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 135 (*Neighbor Solicitation*).

Target (camada ICMPv6)

Este campo contém o endereço IPv6 procurado (2001:db8::10).

6. Para analisar o comportamento dos dispositivos da rede após a tentativa de duplicação de endereços, efetue, em paralelo, os seguintes passos:
 - (a) A coleta dos pacotes trafegados na interface eth0 de n1Original.
 - (b) A coleta dos pacotes trafegados na interface eth0 de n2Duplicate.
 - (c) A verificação de conectividade IPv6 entre n3Host e o endereço IP 2001:db8::10.

As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.

7. Efetue a análise dos pacotes coletados.
- (a) Pacotes coletados da interface eth0 de n1original durante o passo 6. Campos importantes do pacote *echo reply*, representado pela Figura 1.19:

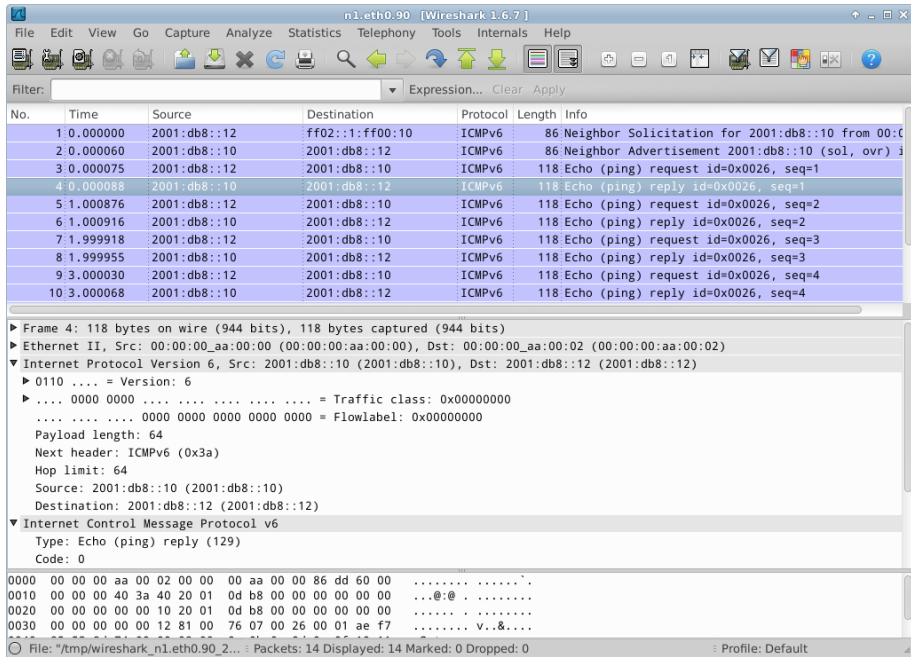


Figura 1.19: pacote *echo reply* capturado em n1original, mostrado no Wireshark.

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se a uma mensagem ICMPv6.

Destination (camada IPv6)

O destino é o endereço IPv6 de n3Host (2001:db8::12).

Source (camada IPv6)

A origem é o endereço IPv6 de n1original (2001:db8::10).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 129 (*echo reply*).

Observe que o dispositivo `n1original` respondeu todas as mensagens *echo request*.

- (b) Pacotes coletados da interface `eth0` de `n2Duplicate` durante o passo 6.

Observe que a interface do endereço duplicado não capturou nenhum pacote destinado ao endereço em questão, somente mensagens destinadas a endereços *multicast*. A Figura 1.20 apresenta uma lista de pacotes coletados em `n2Duplicate`.

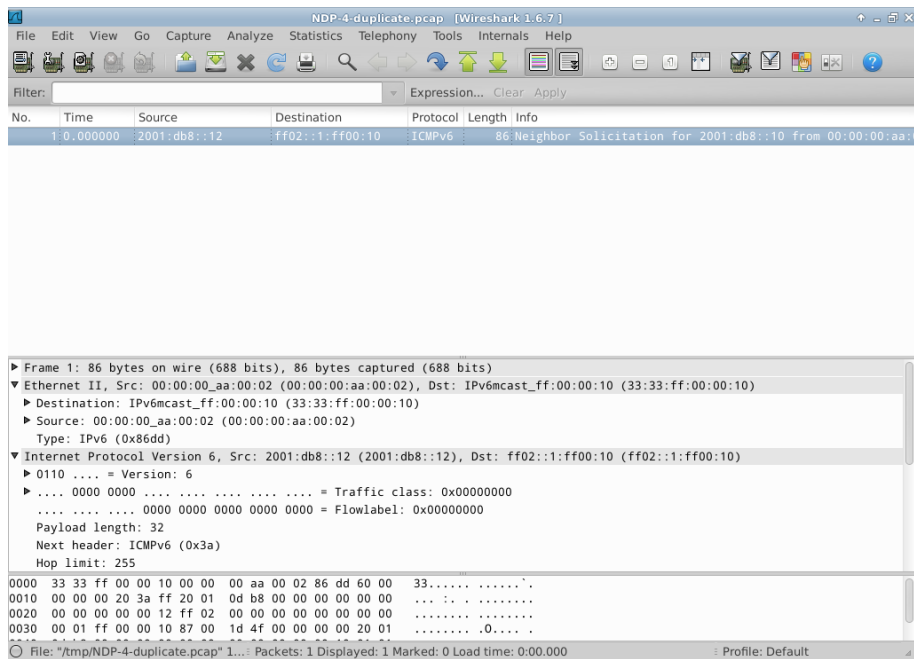


Figura 1.20: lista de pacotes capturados em `n2Duplicate`, mostrada no *Wireshark*.

8. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 1.5. Autoconfiguração *stateless* de endereço: *Router Advertisement* utilizando Quagga

Objetivo

Esta experiência mostra como um nó configura para si um novo endereço, baseando-se em um prefixo enviado pelo roteador. Para isto será configurado o Quagga, uma plataforma de roteamento para servidores Unix, no nó que fará o papel de roteador. Os pacotes RA serão analisados e será observado o endereço configurado automaticamente no outro nó.

Para a realização do presente exercício será utilizada a topologia descrita no arquivo: **1-05-SLAAC-quagga.imn**.

Introdução teórica

A autoconfiguração *stateless* é o processo em que os nós de uma rede podem criar para si próprios endereços, baseados em informações locais e em informações recebidas de roteadores, por meio das mensagens RA. Enquanto o roteador anuncia o prefixo que identifica a rede associada ao enlace, o nó determina o identificador de interface, que o identifica de forma única nessa rede.

Vale notar que também considera-se como parte da autoconfiguração *stateless* a obtenção do endereço *link-local* e a detecção de endereços duplicados para todos os endereços gerados.

A autoconfiguração é dita *stateless* porque o roteador não mantém o registro do estado e características do nó destinatário. Este último encarrega-se de sua própria configuração.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-05-SLAAC-quagga.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 1.21, deve aparecer.

O objetivo dessa topologia de rede é representar o mínimo necessário para que a mensagem RA seja verificada.

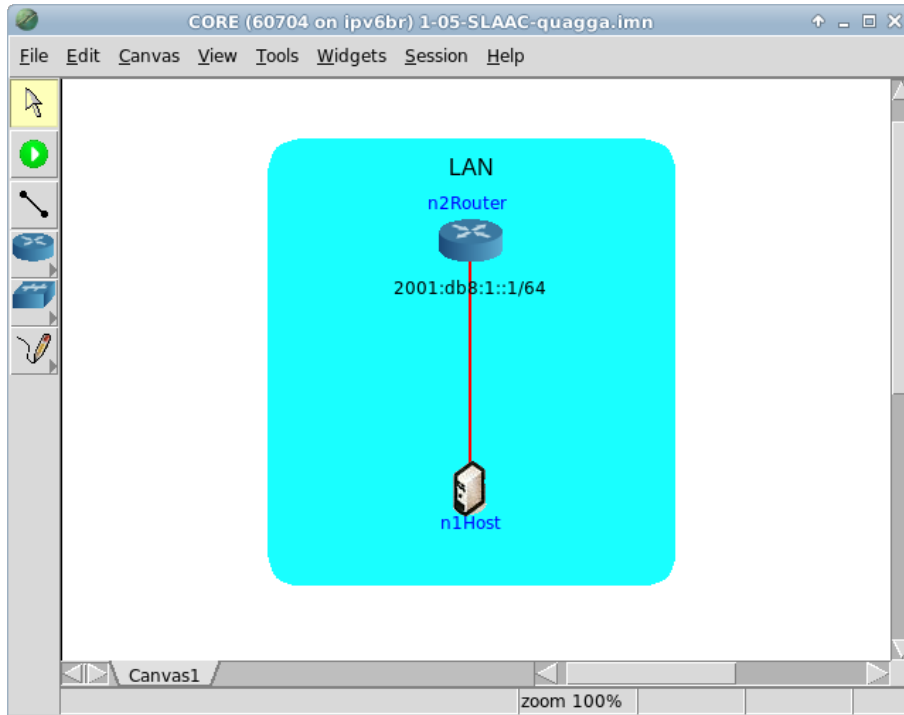


Figura 1.21: topologia da *Experiência 1.5* no CORE.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação e verifique a configuração de endereços IPv6 nos nós n1Host e n2Router.
3. Configure o roteador de modo que o Quagga envie a mensagem RA.
 - (a) Abra um terminal de n2Router com um duplo-clique e edite o arquivo de configuração do Quagga, localizado em `/usr/local/etc/quagga/Quagga.conf`, de modo a adicionar as linhas em **negrito**.

```
interface eth0
  ipv6 address 2001:db8::1/64
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 5
  ipv6 nd prefix 2001:db8::/64
!
```

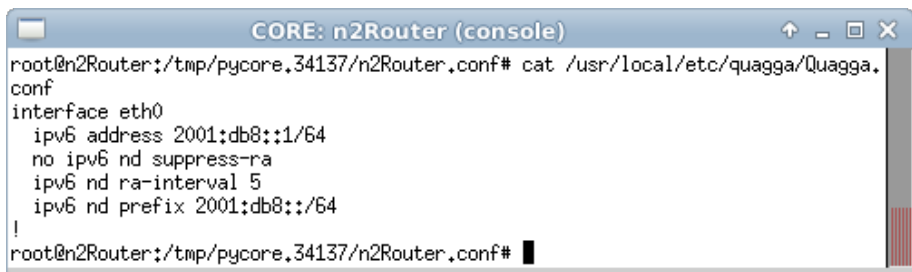
No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano.

É importante observar que na versão 0.99.21mr2.2 do Quagga, a mesma utilizada pelo CORE na VM do IPv6.br, não é possível enviar o endereço do servidor DNS a ser utilizado por meio de RA. Mais informações sobre essa configuração podem ser encontradas em (Ishiguro, 2006).

- (b) Verifique o conteúdo do arquivo de configuração do Quagga. Digite o comando:

```
# cat /usr/local/etc/quagga/Quagga.conf
```

O resultado do comando é representado pela Figura 1.22.



```
root@n2Router:/tmp/pycore.34137/n2Router.conf# cat /usr/local/etc/quagga/Quagga.conf
interface eth0
  ipv6 address 2001:db8::1/64
  no ipv6 nd suppress-ra
  ipv6 nd ra-interval 5
  ipv6 nd prefix 2001:db8::/64
!
root@n2Router:/tmp/pycore.34137/n2Router.conf#
```

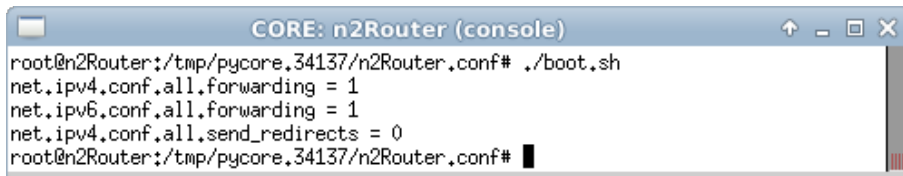
Figura 1.22: *verificação do arquivo de configuração do Quagga após sua edição.*

4. Em paralelo, efetue:
- (a) A coleta dos pacotes trafegados na interface eth0 de n1Host. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.

- (b) Abra um terminal de n2Router com um duplo-clique e utilize o seguinte comando para iniciar o Quagga com as novas configurações:

```
# ./boot.sh
```

Este comando reinicializa os serviços associados ao roteador, incluindo o serviço de roteamento Quagga. Isto é realizado para que o roteador envie a mensagem RA. O resultado do comando é representado pela Figura 1.23.



```
root@n2Router:/tmp/pycore.34137/n2Router.conf# ./boot.sh
net.ipv4.conf.all.forwarding = 1
net.ipv6.conf.all.forwarding = 1
net.ipv4.conf.all.send_redirects = 0
root@n2Router:/tmp/pycore.34137/n2Router.conf# █
```

Figura 1.23: resultado da inicialização do Quagga com a nova configuração.

Após reinicializar o serviço Quagga, verifique a configuração de endereços IPv6 nos nós n1Host e n2Router e a conectividade entre eles, conforme descrito no Apêndice C.

Note que o endereço do nó n1Host originou-se do prefixo de 64 bits enviado pelo roteador.

5. Efetue a análise os pacotes capturados. Aplique o filtro icmpv6 no Wireshark e procure pelos pacotes RA.

Analise os pacotes RA que possuam a opção *Prefix Information* e veja se os dados contidos nos pacotes conferem com a teoria.

Campos importantes do pacote RA, representado pela Figura 1.24:

Destination (camada Ethernet)

O destino é o endereço (33:33:00:00:00:01), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:00:00:01 indica os últimos 32 bits do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do roteador que enviou a resposta (00:00:00:aa:00:00).

```

13 1.000995 fe80::200:ff:feaa:0 ff02::1 ICMPv6 110 Router Advertisement from 00:00:00:aa:00:00
  ▶ Frame 13: 110 bytes on wire (880 bits), 110 bytes captured (880 bits)
  ▶ Ethernet II, Src: 00:00:00_aa:00:00 (00:00:00:aa:00:00), Dst: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
  ▼ Internet Protocol Version 6, Src: fe80::200:ff:feaa:0 (fe80::200:ff:feaa:0), Dst: ff02::1 (ff02::1)
    ▶ 0110 .... = Version: 6
    ▶ .... 0000 0000 .... = Traffic class: 0x00000000
    ▶ .... 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 56
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source: fe80::200:ff:feaa:0 (fe80::200:ff:feaa:0)
    [Source SA MAC: 00:00:00_aa:00:00 (00:00:00:aa:00:00)]
    Destination: ff02::1 (ff02::1)
  ▼ Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0xff75 [correct]
    Cur hop limit: 64
    ▶ Flags: 0x00
    Router lifetime (s): 15
    Reachable time (ms): 0
    Retrans timer (ms): 0
  ▼ ICMPv6 Option (Prefix information : 2001:db8::/64)
    Type: Prefix information (3)
    Length: 4 (32 bytes)
    Prefix Length: 64
    ▶ Flag: 0xc0
    Valid Lifetime: 2592000
    Preferred Lifetime: 604800
    Reserved
    Prefix: 2001:db8:: (2001:db8::)
  ▼ ICMPv6 Option (Source link-layer address : 00:00:00:aa:00:00)
    Type: Source link-layer address (1)
    0040 00 00 00 00 00 00 04 40 c0 00 27 8d 00 00 09 .....@.....
    0050 3a 80 00 00 00 00 20 01 0d b8 00 00 00 00 00 .....:.....
    0060 00 00 00 00 00 00 01 01 00 00 00 aa 00 00 .....
  
```

Figura 1.24: pacote RA mostrado no Wireshark.

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se à uma mensagem ICMPv6.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface que enviou a resposta, sendo neste caso o roteador (fe80::200:ff:feaa:0).

Destination (camada IPv6)

O destino é o endereço *multicast all-nodes* (ff02::1).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 134 (*Router Advertisement*).

ICMPv6 option (camada ICMPv6)

Indica as opções do pacote ICMPv6:

- *Prefix Information*

Type

Contém o valor 3, que identifica o campo *Prefix Information*.

Autonomous Address-Configuration Flag (A)

Contém o valor 1, indicando que o prefixo deve ser utilizado para autoconfiguração *stateless*.

Preferred lifetime

Marca o tempo, em segundos, em que o endereço é preferencial, isto é, o tempo permitido para o uso indistinto do endereço. O valor 0xffffffff indica infinito.

Valid lifetime

Marca o tempo, em segundos, de expiração do endereço gerado. O valor 0xffffffff indica infinito.

Prefix

Contém o prefixo de rede a ser utilizado (2001:db8:).

Prefix length

Contém o tamanho do prefixo da rede.

- *Source link-layer address*

Type

Indica o tipo de opção. Neste caso, *Source link-layer address*.

Link-layer address

Indica o endereço MAC da interface a partir da qual a mensagem de *Router Advertisement* foi enviada, sendo neste caso 00:00:00:aa:00:00.

6. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 1.6. Autoconfiguração *stateless* de endereço: *Router Advertisement* utilizando radvd

Objetivo

Esta experiência mostra como um nó configura para si um novo endereço, a rota padrão e o servidor de DNS, baseando-se em informações enviadas pelo roteador.

Para isto usa-se uma topologia com três nós. Um funcionará como roteador, empregando o *software* radvd, cuja função é enviar as mensagens RA. Um dos nós será o servidor DNS, usando o *software* BIND. No nó cliente o *software* rndssd será usado para habilitar a configuração automática do DNS.

Na experiência será feita a configuração do radvd no roteador e será observada a autoconfiguração do nó cliente.

Para a realização do presente exercício será utilizada a topologia descrita no arquivo: **1-06-SLAAC-radvd.imn**.

Introdução teórica

Veja a introdução teórica da Experiência 1.5.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-06-SLAAC-radvd.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 1.25, deve aparecer.

O objetivo dessa topologia de rede é representar o mínimo necessário para que a mensagem RA seja verificada, juntamente com a funcionalidade de indicação do servidor DNS a ser utilizado pelos clientes do prefixo divulgado.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós n1Host, n2DNS e n3Router e a conectividade entre eles.

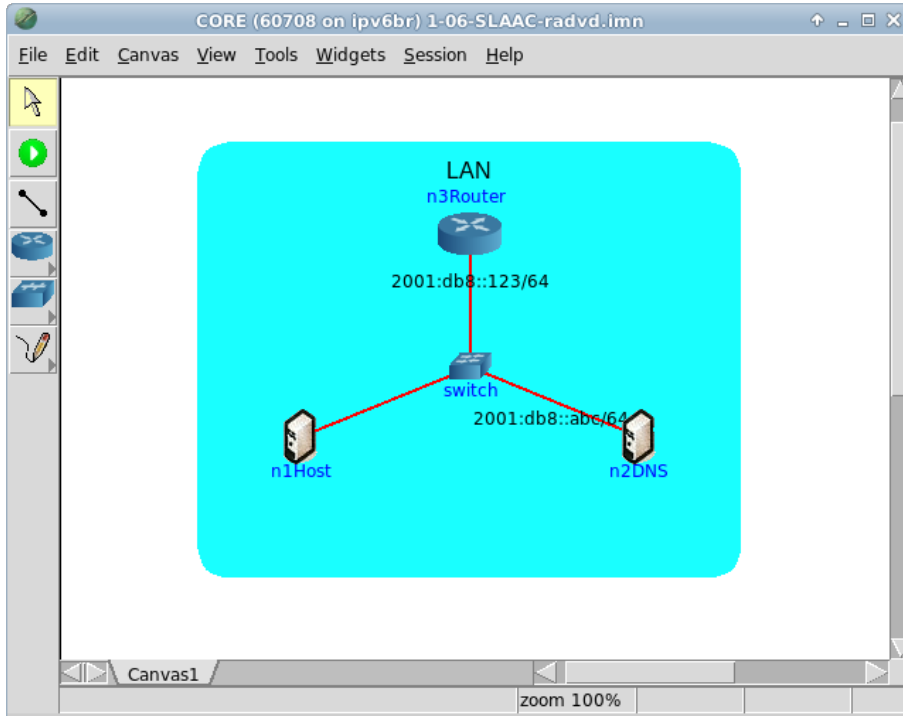


Figura 1.25: topologia da *Experiência 1.6* no CORE.

- Abra um terminal de n2DNS com um duplo-clique e utilize o seguinte comando para iniciar o serviço DNS:

```
# /etc/init.d/bind9 start
```

Este comando inicializa o servidor de DNS BIND na interface eth0 de n2DNS com as configurações padrão. O resultado do comando é representado pela Figura 1.26.

```
n2DNS
root@n2DNS:/tmp/pycore.49958/n2DNS.conf# /etc/init.d/bind9 start
* Starting domain name service... bind9 [ OK ]
root@n2DNS:/tmp/pycore.49958/n2DNS.conf# █
```

Figura 1.26: resultado da inicialização do serviço DNS BIND.

4. Abra um terminal de n1Host com um duplo-clique e utilize o seguinte comando para iniciar o serviço rdnsd:

```
# /etc/init.d/rdnsd start
```

Este comando inicializa o *daemon* de descoberta de servidor DNS recursivo para IPv6 (*IPv6 Recursive DNS Server Discovery Daemon rdnsd*). O resultado do comando é representado pela Figura 1.27.



```
n1Host
root@n1Host:/tmp/pycore.49958/n1Host.conf# /etc/init.d/rdnsd start
* Starting IPv6 Recursive DNS Server discovery Daemon rdnsd [ OK ]
root@n1Host:/tmp/pycore.49958/n1Host.conf# █
```

Figura 1.27: resultado da inicialização do serviço de cliente DNS rdnsd.

5. Configure o roteador de modo que o radvd envie a mensagem RA.
 - (a) Abra um terminal de n3Router com um duplo-clique e crie o arquivo de configuração do radvd, localizado em radvd.conf, com as devidas permissões. Digite os comandos:

```
# touch radvd.conf
# chmod og-w radvd.conf
```

O resultado dos comandos é representado pela Figura 1.28.



```
n3Router
root@n3Router:/tmp/pycore.49958/n3Router.conf# touch radvd.conf
root@n3Router:/tmp/pycore.49958/n3Router.conf# chmod og-w radvd.conf
root@n3Router:/tmp/pycore.49958/n3Router.conf# █
```

Figura 1.28: resultado da criação do arquivo de configuração do radvd.

- (b) Edite o arquivo de configuração do radvd, localizado em radvd.conf:

```
interface eth0 {
    AdvSendAdvert on;
    AdvLinkMTU 1400;
    prefix 2001:db8::/64 {
        AdvOnLink on;
        AdvAutonomous on;
    };
    route ::/0 {};
    RDNSS 2001:db8::abc {};
};
```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano. Para obter mais informações sobre os parâmetros de configuração do radvd consulte (Savola, 2011).

- (c) Verifique o conteúdo do arquivo de configuração do radvd. Digite o seguinte comando:

```
# cat radvd.conf
```

O resultado do comando é representado pela Figura 1.29.

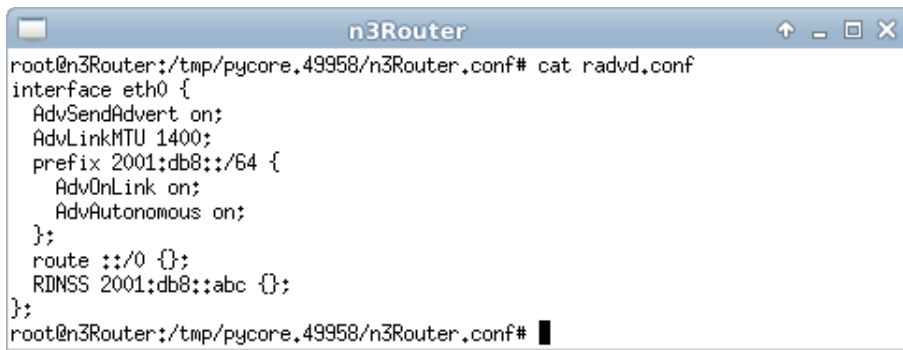
A terminal window titled 'n3Router' with standard window controls (minimize, maximize, close). The prompt is 'root@n3Router:/tmp/pycore.49958/n3Router.conf#'. The user has entered the command 'cat radvd.conf'. The output of the command is the configuration for the 'eth0' interface, including 'AdvSendAdvert on;', 'AdvLinkMTU 1400;', a nested block for 'prefix 2001:db8::/64' with 'AdvOnLink on;' and 'AdvAutonomous on;', 'route ::/0 {};', and 'RDNSS 2001:db8::abc {};', followed by a closing brace and semicolon. The prompt is now 'root@n3Router:/tmp/pycore.49958/n3Router.conf# █'.

Figura 1.29: verificação do arquivo de configuração do radvd após sua edição.

- (d) Valide o conteúdo do arquivo de configuração do radvd. Utilize o seguinte comando:

```
# radvd --configtest -C radvd.conf
```

O resultado do comando é representado pela Figura 1.30.



```
n3Router
root@n3Router:~/tmp/pycore.49958/n3Router.conf# radvd --configtest -C radvd.conf
Syntax OK
root@n3Router:~/tmp/pycore.49958/n3Router.conf# █
```

Figura 1.30: validação do arquivo de configuração do radvd.

6. Em paralelo, efetue:
- A coleta dos pacotes trafegados na interface eth0 de n1Host. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - Efetue os seguintes passos enquanto a coleta é realizada:
 - Abra um terminal de n3Router com um duplo-clique e inicie o serviço radvd. Utilize o seguinte comando:

```
# radvd -C radvd.conf
```

O resultado do comando é representado pela Figura 1.31.



```
n3Router
root@n3Router:~/tmp/pycore.49958/n3Router.conf# radvd -C radvd.conf
root@n3Router:~/tmp/pycore.49958/n3Router.conf# █
```

Figura 1.31: resultado da inicialização do serviço radvd.

Caso a mensagem `daemon: No such device` seja exibida, é seguro ignorá-la, por ser um efeito colateral da virtualização do CORE.

- Abra um terminal de n1Host com um duplo-clique e verifique o recebimento das configurações fornecidas por n3Router com o seguinte comando:

```
# rdisc6 eth0
```

O resultado do comando é representado pela Figura 1.32.

- Ainda no terminal de n1Host, verifique sua configuração de endereços IPv6. Note a presença do prefixo atribuído pelo radvd no endereço obtido por meio da autoconfiguração *stateless*. Caso necessário, consulte o Apêndice C para verificar o procedimento.

```

n1Host
root@n1Host:/tmp/pycore.49958/n1Host.conf# rdisc6 eth0
Soliciting ff02::2 (ff02::2) on eth0...

Hop limit           :           64 (    0x40)
Stateful address conf. :           No
Stateful other conf. :           No
Router preference   :           medium
Router lifetime     :           1800 (0x00000708) seconds
Reachable time      :   unspecified (0x00000000)
Retransmit time     :   unspecified (0x00000000)
Prefix              : 2001:db8::/64
  Valid time        :           86400 (0x00015180) seconds
  Pref. time        :           14400 (0x00003840) seconds
Route               : ::/0
  Route preference  :           medium
  Route lifetime    :           1800 (0x00000708) seconds
Recursive DNS server : 2001:db8::abc
DNS server lifetime :           600 (0x00000258) seconds
MTU                 :           1400 bytes (valid)
Source link-layer address: 00:00:00:AA:00:02
from fe80::200:ff:feaa:2
root@n1Host:/tmp/pycore.49958/n1Host.conf# █

```

Figura 1.32: resultado do recebimento das configurações fornecidas a n1Host.

- iv. Ainda no terminal de n1Host, valide a configuração obtida para o uso de servidor DNS. Utilize o seguinte comando:

```
# dig -x 2001:db8::
```

O resultado do comando é representado pela Figura 1.33.

- v. Ainda no terminal de n1Host, verifique suas rotas IPv6. Para isto utilize o comando:

```
# ip -6 route list
```

O resultado do comando é representado pela Figura 1.34.

- vi. Abra um terminal de n2DNS com um duplo-clique e verifique sua conectividade IPv6 com n1Host. Utilize o seguinte comando:

```
# ping6 -c 4 2001:db8::200:ff:feaa:0
```

O resultado do comando é representado pela Figura 1.35.

7. Efetue a análise os pacotes capturados. Aplique o filtro icmpv6 no Wireshark e procure pelos pacotes RA.

Analise os pacotes RA que possuam a opção *Prefix Information* e veja se os dados contidos nos pacotes conferem com a teoria.

Campos importantes do pacote RA, representado pela Figura 1.36:

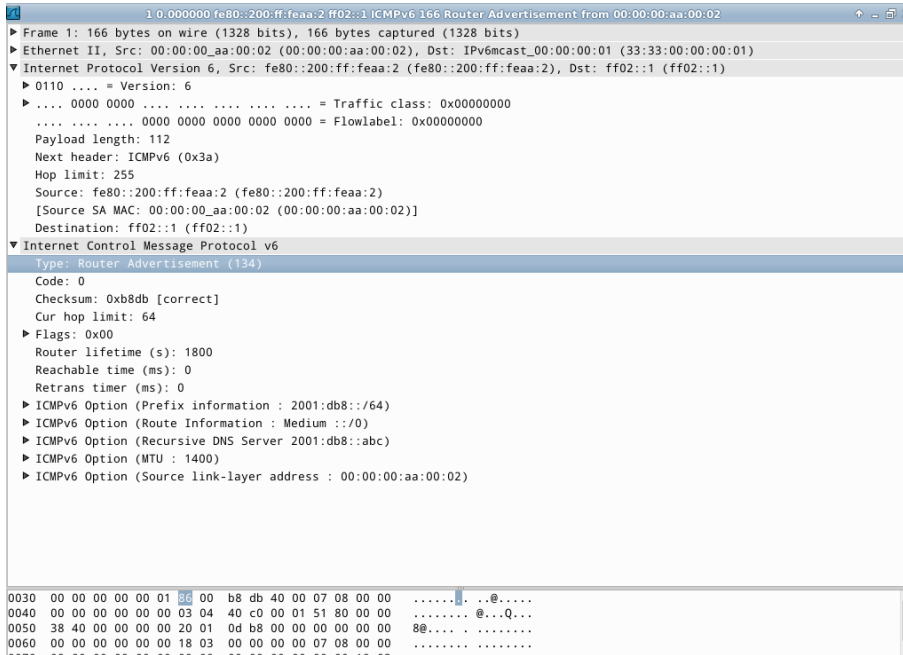


Figura 1.36: pacote RA mostrado no Wireshark.

Destination (camada Ethernet)

O destino é o endereço (33:33:00:00:00:01), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:00:00:01 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do roteador que enviou a resposta (00:00:00:aa:00:02).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se a uma mensagem ICMPv6.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface que enviou a resposta, sendo neste caso o roteador (fe80::200:ff:feaa:2).

Destination (camada IPv6)

O destino é o endereço *multicast all-nodes* (ff02::1).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 134 (*Router Advertisement*).

ICMPv6 option (camada ICMPv6)

Indica as opções do pacote ICMPv6:

- *Prefix Information*

Type

Contém o valor 3, que identifica o campo *Prefix Information*.

Autonomous Address-Configuration Flag (A)

Contém o valor 1, indicando que o prefixo deve ser utilizado para autoconfiguração *stateless*.

Preferred lifetime

Marca o tempo, em segundos, em que o endereço é preferencial, isto é, tempo permitido para o uso indistinto do endereço. O valor 0xffffffff indica infinito.

Valid lifetime

Marca o tempo, em segundos, de expiração do endereço gerado. O valor 0xffffffff indica infinito.

Prefix

Contém o prefixo de rede a ser utilizado (2001:db8::).

Prefix length

Contém o tamanho do prefixo da rede.

- *Source link-layer address*

Type

Indica o tipo de opção. Neste caso, *Source link-layer address*.

Link-layer address

Indica o endereço MAC da interface a partir da qual a mensagem de *Router Advertisement* foi enviada, sendo neste caso 00:00:00:aa:00:02.

- *Route Information*

Type

Indica o tipo de opção. Neste caso, *Route Information*.

Prefix

Indica um prefixo alcançável pelo roteador que divulgou a mensagem (::/0).

Length

Indica o tamanho do prefixo divulgado na mensagem.

- *Recursive DNS Server*

Type

Indica o tipo de opção. Neste caso, *Recursive DNS Server*.

Recursive DNS Server

Indica o endereço IPv6 do servidor DNS a ser utilizado (2001:db8::abc).

- *MTU*

Type

Indica o tipo de opção. Neste caso, *MTU*.

MTU

Indica o tamanho máximo de *MTU* a ser utilizado no enlace em questão, sendo 1400 neste caso.

8. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 1.7. DHCPv6 *stateful*: *Solicit*, *Advertise*, *Request* e *Reply*

Objetivo

Esta experiência apresenta o funcionamento do DHCPv6 no modo *stateful*. O servidor DHCPv6 será configurado para informar o endereço IPv6 que o cliente deverá atribuir a sua interface de rede, registrando qual cliente recebeu cada endereço. Além do endereço IPv6, o endereço do servidor DNS recursivo da rede também será informado.

O dispositivo n1Host será o cliente que realizará as requisições e o n3DNS representa o servidor DNS recursivo da rede, utilizado para validar se as informações divulgadas pelo servidor DHCPv6 foram aprendidas corretamente.

Para a realização do presente exercício, será utilizada a topologia descrita no arquivo: **1-07-DHCPv6-stateful.imn**.

Introdução teórica

O *Dynamic Host Configuration Protocol* (DHCP) é um protocolo utilizado para distribuir dinamicamente endereços IP e parâmetros de configuração da rede. Há implementações específicas para redes IPv6 e IPv4, que apresentam as mesmas funcionalidades básicas, porém, com significativas diferenças nas opções que podem ser enviadas. Nesta introdução, apenas as características do DHCPv6 são tratadas.

Basicamente, a comunicação entre o servidor DHCP e as máquinas cliente se dá com a troca de quatro mensagens:

- **Solicit:** enviada pelo cliente ao grupo *multicast all-dhcp-agents* (ff02::1:2) com o intuito de localizar o servidor DHCP.
- **Advertise:** enviada pelo servidor DHCP, diretamente ao endereço *link-local* do cliente, para indicar que ele pode fornecer as informações necessárias para a configuração.
- **Request:** enviada pelo cliente diretamente ao grupo *multicast all-dhcp-agents* (ff02::1:2) para requisitar ao servidor DHCP os dados de configuração.
- **Reply:** enviada pelo servidor DHCP ao endereço de *link-local* do cliente como resposta à mensagem *Request*.

O DHCPv6 possui dois modos de operação:

- **Stateful:** o servidor DHCPv6 é responsável por informar aos clientes os endereços IPv6 que devem ser utilizados em suas interfaces de rede, mantendo o estado de qual endereço foi atribuído a determinado cliente.
- **Stateless:** o servidor DHCPv6 informa apenas parâmetros de configuração como endereço dos servidores DNS ou servidores SIP da rede aos clientes, sem a necessidade de guardar qual informação individual de cada cliente. Nesse segundo caso, o cliente deverá obter o endereço IPv6 de sua interface de outra forma, seja manualmente ou SLAAC.

Diferentemente do DHCPv4, o DHCPv6 não possui a opção **Default routers**, que informa o endereço do roteador padrão da rede. Deste modo, para que o cliente obtenha conectividade com outras redes, é preciso utilizar o DHCPv6 em conjunto com o protocolo *Neighbor Discovery*, ou realizar a configuração do roteador padrão manualmente em cada cliente. Em outras palavras, o DHCPv6 normalmente é utilizado como um complemento da autoconfiguração *stateless*, seja utilizado em modo *stateful* ou *stateless*.

Para o uso do DHCPv6 em conjunto com o NDP é necessário habilitar o envio das mensagens RA nos roteadores da rede, para que estes se anunciem como roteadores padrão e manipular duas *flags* das mensagens RA de acordo com os parâmetros que os clientes deverão aprender do servidor DHCPv6: (i) *Managed address configuration flag*, que define a permissão do recebimento do endereço IPv6, por meio do servidor DHCPv6; e (ii) *Other configuration flag*, que habilita o recebimento de outras configurações vindas do servidor DHCPv6.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-07-DHCPv6-stateful.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 1.37, deve aparecer.

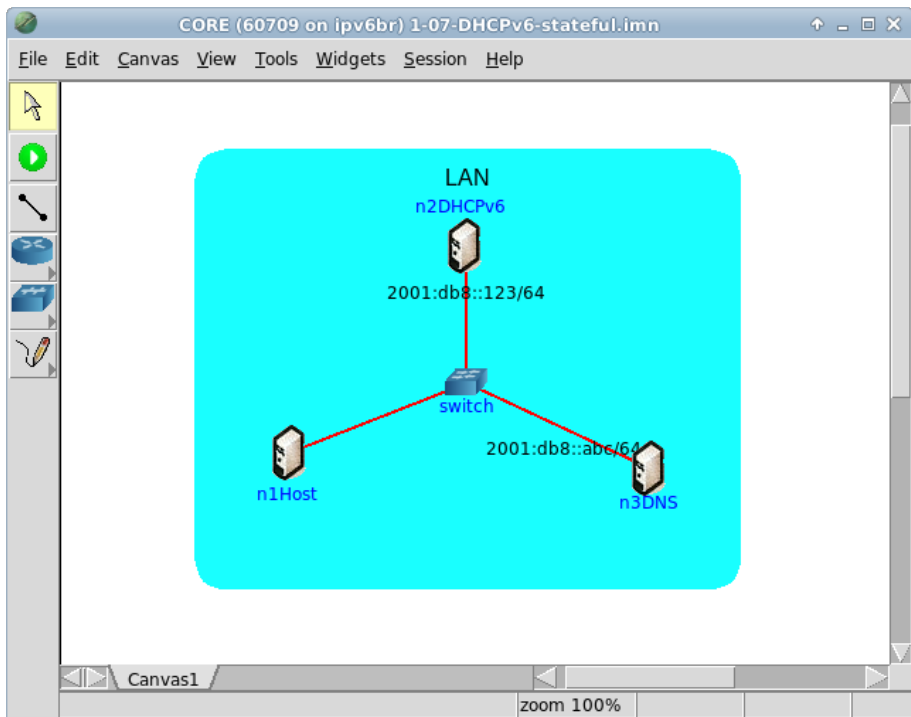


Figura 1.37: topologia da *Experiência 1.7* no CORE.

O objetivo dessa topologia de rede é representar o mínimo necessário para que a troca de mensagens DHCPv6 seja verificada, juntamente com a funcionalidade de indicação do servidor DNS a ser utilizado pelos clientes.

Note que não há um roteador nessa topologia e que o DHCP no IPv6 não informa o roteador padrão para os clientes. Então, os nós não terão uma rota padrão.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós n1Host, n2DHCPv6 e n3DNS e a conectividade entre eles.
3. Abra um terminal de n3DNS com um duplo-clique e utilize o seguinte comando para iniciar o serviço DNS:

```
# /etc/init.d/bind9 start
```

Este comando inicializa o servidor de DNS BIND na interface eth0 de servidor DNS com suas configurações padrão. O resultado do comando é representado pela Figura 1.38.




Figura 1.38: resultado da inicialização do serviço DNS BIND.

4. Configure o serviço DHCPv6 para enviar as configurações ao cliente.
 - (a) Abra um terminal de n2DHCPv6 e crie os arquivos de configuração do DHCPv6. Para isto digite os seguintes comandos:

```
# touch dhcpd.conf  
# touch dhcpd.leases
```

O resultado dos comandos é representado pela Figura 1.39.



```

CORE: n2DHCPv6 (console)
root@n2DHCPv6:/tmp/pycore.34143/n2DHCPv6.conf# touch dhcpd.conf
root@n2DHCPv6:/tmp/pycore.34143/n2DHCPv6.conf# touch dhcpd.leases
root@n2DHCPv6:/tmp/pycore.34143/n2DHCPv6.conf# █

```

Figura 1.39: resultado da criação dos arquivos de configuração do DHCPv6.

- (b) Ainda no terminal de n2DHCPv6, edite o arquivo de configuração do DHCPv6, localizado em `dhcpd.conf`, de modo a inserir as seguintes linhas:

```

default-lease-time 600;
max-lease-time 7200;
subnet6 2001:db8::/64 {
    range6 2001:db8::1234 2001:db8::abcd;
    option dhcp6.name-servers 2001:db8::abc;
}

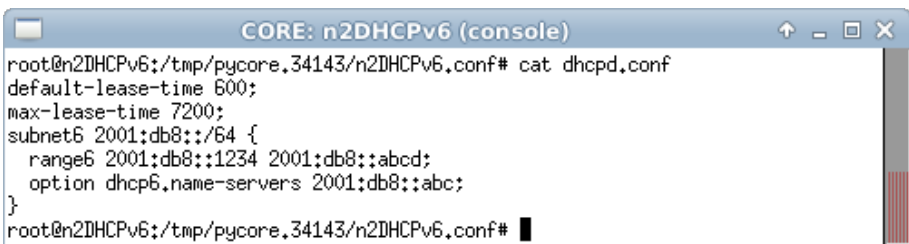
```

O campo `name-servers` contém o endereço da máquina que funcionará como servidor DNS. O campo `range6` contém a faixa de endereços dentro do prefixo de sub-rede configurado, que será distribuída entre os dispositivos clientes. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

- (c) Verifique o conteúdo do arquivo de configuração do DHCPv6. Digite o seguinte comando:

```
# cat dhcpd.conf
```

O resultado do comando é representado pela Figura 1.40.



```

CORE: n2DHCPv6 (console)
root@n2DHCPv6:/tmp/pycore.34143/n2DHCPv6.conf# cat dhcpd.conf
default-lease-time 600;
max-lease-time 7200;
subnet6 2001:db8::/64 {
    range6 2001:db8::1234 2001:db8::abcd;
    option dhcp6.name-servers 2001:db8::abc;
}
root@n2DHCPv6:/tmp/pycore.34143/n2DHCPv6.conf# █

```

Figura 1.40: verificação do arquivo de configuração do DHCPv6 após sua edição.

(d) Inicie o serviço DHCPv6 com o comando:

```
# dhcpcd -6 -cf dhcpcd.conf -lf dhcpcd.leases
```

O resultado do comando é representado pela Figura 1.41.



```

CORE: n2DHCPv6 (console)
root@n2DHCPv6:/tmp/pycore.34143/n2DHCPv6.conf# dhcpcd -6 -cf dhcpcd.conf -lf dhcpcd
.leases
Internet Systems Consortium DHCP Server 4.2.4-P2
Copyright 2004-2012 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Wrote 0 leases to leases file.
Bound to *:547
Listening on Socket/5/eth0/2001:db8::/64
Sending on Socket/5/eth0/2001:db8::/64
root@n2DHCPv6:/tmp/pycore.34143/n2DHCPv6.conf#

```

Figura 1.41: resultado da inicialização do serviço DHCPv6.

5. Em paralelo, efetue:

(a) A coleta dos pacotes trafegados na interface eth0 den1Host. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.

(b) Efetue os seguintes passos enquanto a coleta é realizada:

i. Abra um terminal de n1Host com um duplo-clique e inicie o serviço dhclient. Para isto digite os comandos:

```
# touch dhclient6.conf
# touch dhclient6.leases
# dhclient -6 -cf dhclient6.conf -lf dhclient6.leases
```

O resultado dos comandos é representado pela Figura 1.42.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.34143/n1Host.conf# touch dhclient6.conf
root@n1Host:/tmp/pycore.34143/n1Host.conf# touch dhclient6.leases
root@n1Host:/tmp/pycore.34143/n1Host.conf# dhclient -6 -cf dhclient6.conf -lf dh
client6.leases
root@n1Host:/tmp/pycore.34143/n1Host.conf#

```

Figura 1.42: resultado da inicialização do serviço dhclient.

Pode-se observar que o arquivo de configuração dhclient6.conf encontra-se vazio. A configuração padrão do cliente DHCPv6 dhclient efetua a requisição de associação de identidade (*identity-association* – IA), servidores DNS a serem utilizados e a lista de busca de domínio.

- ii. Ainda no terminal de n1Host, visualize as configuração adquiridas. Utilize os seguintes comandos:

```
# ip addr show
# cat /etc/resolv.conf
```

O resultado dos comandos é representado pela Figura 1.43. Note o endereço IPv6 obtido.



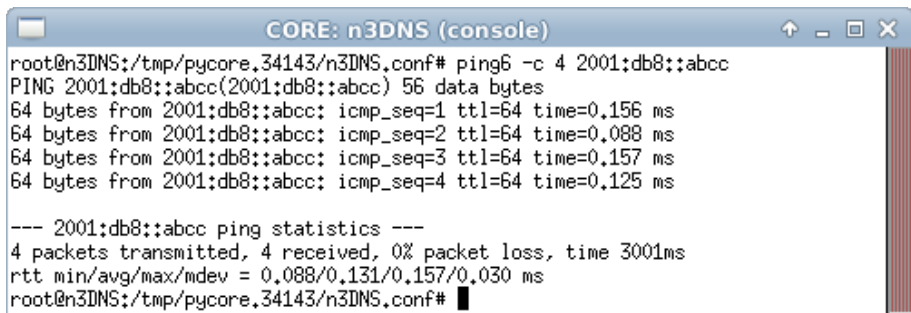
```
root@n1Host:/tmp/pycore.34143/n1Host.conf# dhclient -6 -cf dhclient6.conf -lf dhclient6.leases
root@n1Host:/tmp/pycore.34143/n1Host.conf# ip addr show
60: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
64: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
    inet6 2001:db8::abcc/64 scope global
        valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feaa:0/64 scope link
        valid_lft forever preferred_lft forever
root@n1Host:/tmp/pycore.34143/n1Host.conf# cat /etc/resolv.conf
nameserver 2001:db8::abc
root@n1Host:/tmp/pycore.34143/n1Host.conf#
```

Figura 1.43: resultado das configurações adquiridas por meio do DHCPv6.

- iii. Abra um terminal de n3DNS e verifique a conectividade com n1Host. Digite o seguinte comando:

```
# ping6 -c 4 2001:db8::abcc
```

O resultado do comando é representado pela Figura 1.44.



```
root@n3DNS:/tmp/pycore.34143/n3DNS.conf# ping6 -c 4 2001:db8::abcc
PING 2001:db8::abcc(2001:db8::abcc) 56 data bytes
64 bytes from 2001:db8::abcc: icmp_seq=1 ttl=64 time=0,156 ms
64 bytes from 2001:db8::abcc: icmp_seq=2 ttl=64 time=0,088 ms
64 bytes from 2001:db8::abcc: icmp_seq=3 ttl=64 time=0,157 ms
64 bytes from 2001:db8::abcc: icmp_seq=4 ttl=64 time=0,125 ms

--- 2001:db8::abcc ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0,088/0,131/0,157/0,030 ms
root@n3DNS:/tmp/pycore.34143/n3DNS.conf#
```

Figura 1.44: verificação da conectividade entre n3DNS e n1Host.


```

1 0.000000 fe80::200:ff:feaa:0:ff02::1:2 DHCPv6 114 Solicit XID: 0xcfec30 CID: 00010001182d97b4000000aa0000
  ▶ Frame 1: 114 bytes on wire (912 bits), 114 bytes captured (912 bits)
  ▶ Ethernet II, Src: 00:00:00_aa:00:00 (00:00:00:aa:00:00), Dst: IPv6mcast_00:01:00:02 (33:33:00:01:00:02)
  ▼ Internet Protocol Version 6, Src: fe80::200:ff:feaa:0 (fe80::200:ff:feaa:0), Dst: ff02::1:2 (ff02::1:2)
    ▶ 0110 .... = Version: 6
    ▶ .... 0000 0000 .... = Traffic class: 0x00000000
    ▶ .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 60
    Next header: UDP (0x11)
    Hop limit: 1
    Source: fe80::200:ff:feaa:0 (fe80::200:ff:feaa:0)
    [Source SA MAC: 00:00:00_aa:00:00 (00:00:00:aa:00:00)]
    Destination: ff02::1:2 (ff02::1:2)
  ▼ User Datagram Protocol, Src Port: dhcpv6-client (546), Dst Port: dhcpv6-server (547)
    Source port: dhcpv6-client (546)
    Destination port: dhcpv6-server (547)
    Length: 60
    ▶ Checksum: 0x3940 [validation disabled]
  ▼ DHCPv6
    Message type: Solicit (1)
    Transaction ID: 0xcfec30
    ▶ Client Identifier: 00010001182d97b4000000aa0000
    ▶ Option Request
    ▶ Elapsed time
    ▶ Identity Association for Non-temporary Address
  
```

```

0030 00 00 00 01 00 02 02 22 02 23 00 3c 39 40 01 cf .....".#.-9@.
0040 ec 30 00 01 00 0e 00 01 00 01 18 2d 97 b4 00 00 .0.....
0050 00 aa 00 00 00 06 00 04 00 17 00 18 00 08 00 02 .....
0060 00 00 00 03 00 0c 00 aa 00 00 00 00 0e 10 00 00 .....
  
```

Figura 1.46: pacote *Solicit* mostrado no *Wireshark*.

Destination (camada Ethernet)

O destino é o endereço (33:33:00:01:00:02), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:01:00:02 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do dispositivo que enviou a solicitação (00:00:00:aa:00:00).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface diretamente conectada ao enlace em que se fez a requisição (fe80::200:ff:feaa:0).

Destination (camada IPv6)

O destino é o endereço *multicast all-dhcp-agents* (ff02::1:2).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Message type (camada DHCPv6)

Indica por meio do valor 1 que o tipo de mensagem é *Solicit*.

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Identity association for non-temporary address (camada DHCPv6)

Identifica a requisição de endereço IPv6 para o servidor.

Option request (camada DHCPv6)

Indica as opções do pacote DHCPv6.

Requested option code

Indica a solicitação ao servidor DHCPv6 do DNS *Recursive Name Server*, por meio do valor 23 e a solicitação da lista de busca de domínio, pelo valor 24.

Campos importantes do pacote *Advertise*, representado pela Figura 1.47:

Destination (camada Ethernet)

O destino é o endereço MAC da máquina solicitante (00:00:00:aa:00:00).

Source (camada Ethernet)

A origem é o endereço MAC da interface da máquina que enviou a resposta (00:00:00:aa:00:01).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

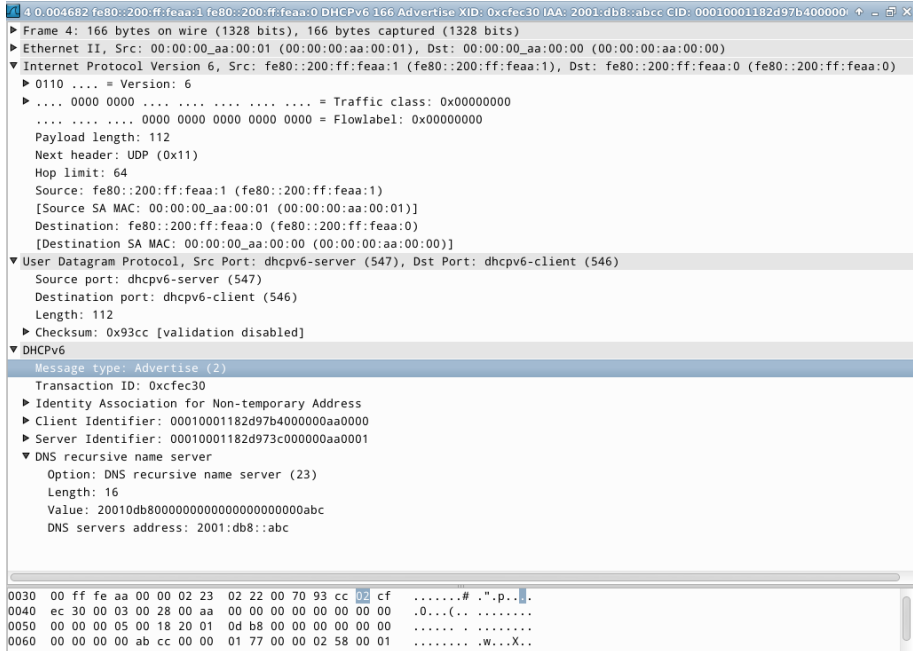


Figura 1.47: pacote *Advertise* mostrado no Wireshark.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface do dispositivo que enviou a mensagem, ou seja, do servidor DHCPv6 (fe80::200:ff:feaa:1).

Destination (camada IPv6)

O destino é o endereço *unicast* de *link-local* da máquina solicitante (fe80::200:ff:feaa:0).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Message type (camada DHCPv6)

Indica por meio do valor 2 que o tipo de mensagem é *Advertise*.

Identity association for non-temporary address (camada DHCPv6)

Identifica o uso do endereço IPv6 entregue para o cliente.

IA Address

Contém o endereço e as configurações que o cliente deve utilizar (2001:db8::abcc).

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Server identifier (camada DHCPv6)

Contém dados da identificação única do servidor baseada no endereço físico.

DNS recursive name server (camada DHCPv6)

Indica as informações relacionadas aos servidores DNS recursivos.

DNS servers address

Indica o endereço IPv6 do servidor DNS requisitado (2001:db8::abc).

```

5 1.051183 fe80::200:ff:feaa:0:ff02::1:2 DHCPv6 160 Request ID: 0x8433be CID: 00010001182d97b4000000aa0000 IAA: 2001:db8::abcc
▶ Frame 5: 160 bytes on wire (1280 bits), 160 bytes captured (1280 bits)
▶ Ethernet II, Src: 00:00:00_aa:00:00 (00:00:00:aa:00:00), Dst: IPv6mcast_00:01:00:02 (33:33:00:01:00:02)
▼ Internet Protocol Version 6, Src: fe80::200:ff:feaa:0 (fe80::200:ff:feaa:0), Dst: ff02::1:2 (ff02::1:2)
  ▶ 0110 .... = Version: 6
  ▶ .... 0000 0000 .... = Traffic class: 0x00000000
  .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
  Payload length: 106
  Next header: UDP (0x11)
  Hop limit: 1
  Source: fe80::200:ff:feaa:0 (fe80::200:ff:feaa:0)
  [Source SA MAC: 00:00:00_aa:00:00 (00:00:00:aa:00:00)]
  Destination: ff02::1:2 (ff02::1:2)
▼ User Datagram Protocol, Src Port: dhcpv6-client (546), Dst Port: dhcpv6-server (547)
  Source port: dhcpv6-client (546)
  Destination port: dhcpv6-server (547)
  Length: 106
  ▶ Checksum: 0x2c50 [validation disabled]
▼ DHCPv6
  Message type: Request (3)
  Transaction ID: 0x8433be
  ▶ Client Identifier: 00010001182d97b4000000aa0000
  ▶ Server Identifier: 00010001182d973c000000aa0001
  ▼ Option Request
    Option: Option Request (6)
    Length: 4
    Value: 00170018
    Requested Option code: DNS recursive name server (23)
    Requested Option code: Domain Search List (24)
  ▶ Elapsed time
  ▶ Identity Association for Non-temporary Address
0030 00 00 00 01 00 02 02 22 02 23 00 6a 2c 50 06 84 .....".#.,P.
0040 33 be 00 01 00 0e 00 01 00 01 18 2d 97 b4 00 00 3.....
0050 00 aa 00 00 00 02 00 0e 00 01 00 01 18 2d 97 3c .....<
0060 00 00 00 aa 00 01 00 06 00 04 00 17 00 18 00 08 .....

```

Figura 1.48: pacote *Request* mostrado no Wireshark.

Campos importantes do pacote *Request*, representado pela Figura 1.48:

Destination (camada Ethernet)

O destino é o endereço (33:33:00:01:00:02), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:01:00:02 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do dispositivo que enviou a requisição (00:00:00:aa:00:00).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface do dispositivo que enviou a mensagem, ou seja, do cliente (fe80::200:ff:feaa:0).

Destination (camada IPv6)

O destino é o endereço *multicast all-dhcp-agents* DHCPv6 (ff02::1:2).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Message type (camada DHCPv6)

Indica por meio do valor 3 que o tipo de mensagem é *Request*.

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Identity association for non-temporary address (camada DHCPv6)

Identifica a confirmação do recebimento do endereço IPv6 para o servidor:

IA Address

Contém o endereço e as configurações que o cliente deve utilizar (2001:db8::abcc).

Option request (camada DHCPv6)

Indica as opções do pacote DHCPv6:

Requested option code

Indica a solicitação ao servidor DHCPv6 do DNS *Recursive Name Server*, por meio do valor 23 e a solicitação da lista de busca de domínio, pelo valor 24.

Server identifier (camada DHCPv6)

Contém dados da identificação única do servidor baseada no endereço físico.

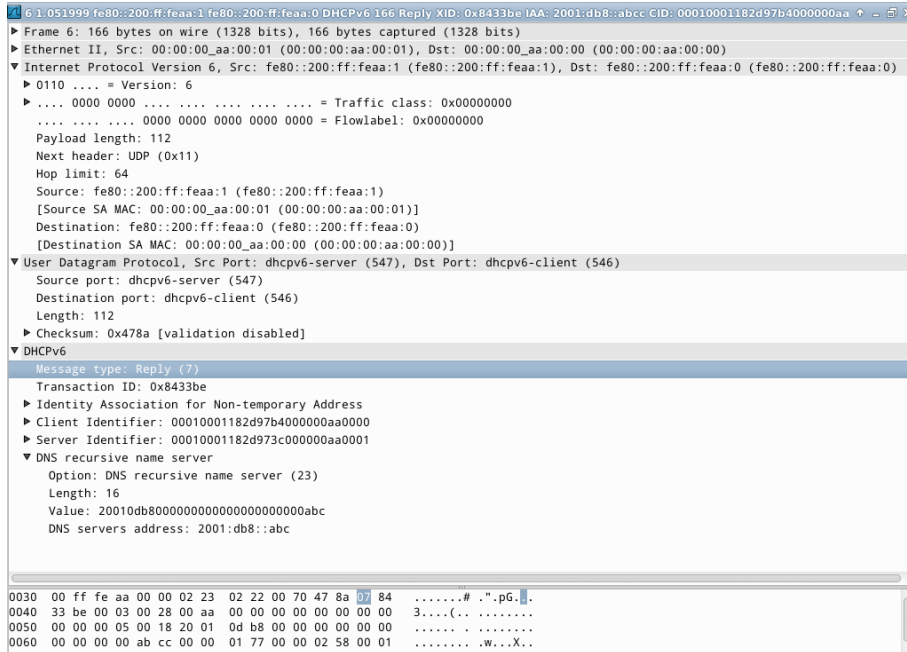


Figura 1.49: pacote Reply mostrado no Wireshark.

Campos importantes do pacote *Reply*, representado pela Figura 1.49:

Destination (camada Ethernet)

O destino é o endereço MAC da máquina solicitante (00:00:00:aa:00:00).

Source (camada Ethernet)

A origem é o endereço MAC da interface da máquina que enviou a resposta (00:00:00:aa:00:01).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface do dispositivo que enviou a mensagem, ou seja, do servidor DHCPv6 (fe80::200:ff:feaa:1).

Destination (camada IPv6)

O destino é o endereço IPv6 *unicast* de *link-local* do cliente (fe80::200:ff:feaa:0).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Message type (camada DHCPv6)

Indica por meio do valor 7 que o tipo de mensagem é *Reply*.

Identity association for non-temporary address (camada DHCPv6)

Identifica a confirmação do fornecimento do endereço IPv6 para o cliente.

IA Address

Contém o endereço e as configurações que o cliente deve utilizar (2001:db8::abcc).

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Server identifier (camada DHCPv6)

Contém dados da identificação única do servidor baseada no endereço físico.

DNS recursive name server (camada DHCPv6)

Indica as informações relacionadas aos servidores DNS recursivos.

DNS servers address

Indica o endereço IPv6 do servidor DNS requisitado (2001:db8::abc).

7. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 1.8. DHCPv6 *stateless: Information-Request e Reply*

Objetivo

Esta experiência apresenta o funcionamento do DHCPv6 no modo *stateless*, em que o servidor informa apenas parâmetros de configuração que não implicam na necessidade da guarda de qualquer informação individual dos clientes.

Inicialmente, o `n4Router` será configurado para divulgar por meio da mensagem RA o prefixo Global IPv6 da rede e a *flag Other Configuration*, de modo que o cliente `n1Host` possa gerar o endereço de sua interface utilizando SLAAC e requisitar as demais configurações de rede ao servidor DHCPv6.

Nos passos seguintes, o servidor `n2DHCPv6` será configurado para informar ao nó `n1Host` o endereço do servidor DNS, que realizará uma consulta ao `n3DNS` para validar a informação aprendida.

Para a realização do presente exercício, será utilizada a topologia descrita no arquivo: **1-08-DHCPv6-stateless.imn**.

Introdução teórica

Vide a introdução teórica da experiência anterior: Experiência 1.7

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-08-DHCPv6-stateless.imn** localizado no diretório `lab`, dentro do `Desktop`. A topologia de rede, representada pela Figura 1.50, deve aparecer. `WIDE-DHCPv6 client`

O objetivo dessa topologia de rede é representar o mínimo necessário para que a troca de mensagens DHCPv6 seja verificada, juntamente com a funcionalidade de indicação do servidor DNS a ser utilizado pelos clientes.

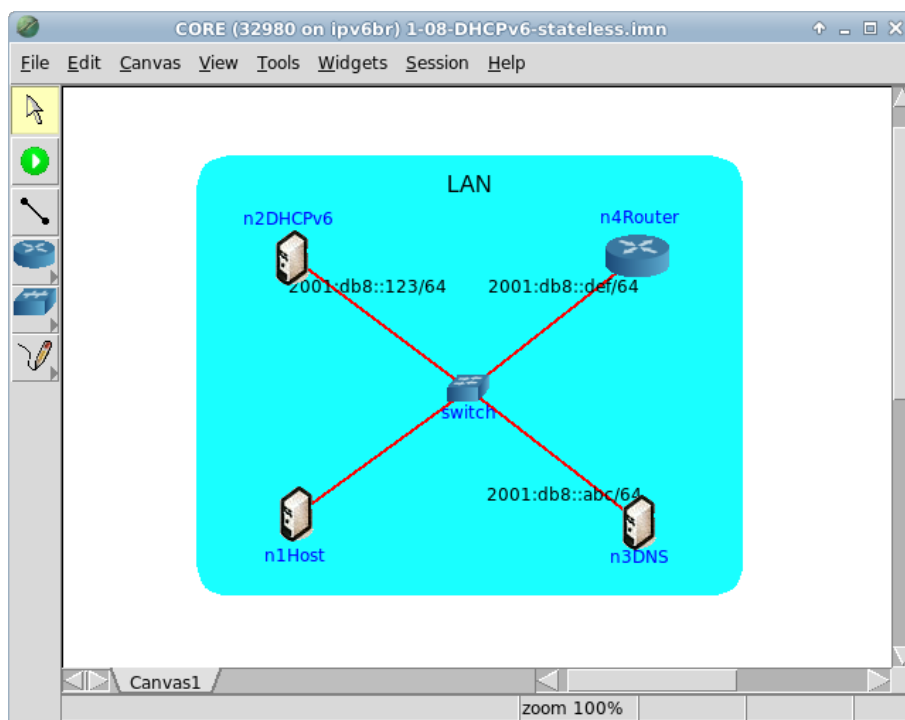


Figura 1.50: topologia da *Experiência 1.8* no CORE.

2. Conforme descrito nos Apêndices B e C, inicie a simulação, verifique a configuração de endereços IPv6 nos nós n1Host, n2Dhcpv6, n3DNS, n4Router e a conectividade entre eles.
3. Abra um terminal de n3DNS e utilize o seguinte comando para iniciar o serviço DNS:

```
# /etc/init.d/bind9 start
```

Este comando inicializa o servidor de DNS BIND na interface eth0 de n2DNS com as configurações padrão. O resultado do comando é representado pela Figura 1.51.

```
CORE: n3DNS (console)
root@n3DNS:~/tmp/pycore,.58550/n3DNS.conf# /etc/init.d/bind9 start
* Starting domain name service... bind9
root@n3DNS:~/tmp/pycore,.58550/n3DNS.conf#
```

Figura 1.51: resultado da inicialização do serviço DNS BIND.

4. Configure o roteador de modo que o serviço de roteamento Quagga envie a mensagem RA para que o nó n1Host efetue a autoconfiguração *stateless*.
- (a) Abra um terminal de n4Router com um duplo-clique e edite o arquivo de configuração do Quagga, localizado em `/usr/local/etc/quagga/Quagga.conf`, de modo a adicionar as linhas em **negrito**:

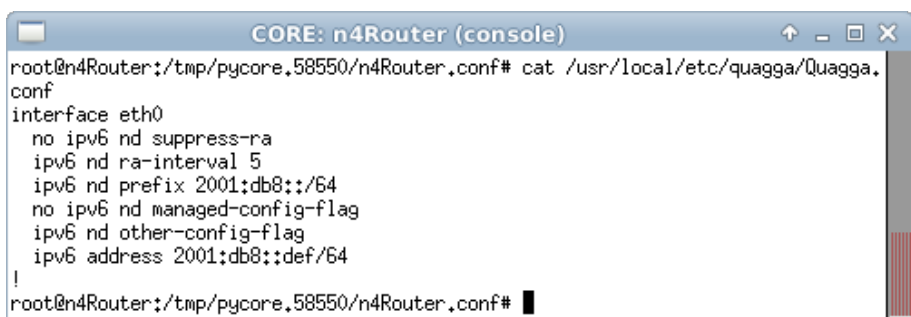
```
interface eth0
no ipv6 nd suppress-ra
ipv6 nd ra-interval 5
ipv6 nd prefix 2001:db8::/64
no ipv6 nd managed-config-flag
ipv6 nd other-config-flag
ipv6 address 2001:db8::def/64
!
```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano. Até a versão 0.99.21mr2.2 do Quagga, a mesma utilizada pelo CORE na VM do IPv6.br, não é possível enviar o endereço do servidor DNS recursivo por meio de RA.

- (b) Verifique o conteúdo do arquivo de configuração do Quagga:

```
# cat /usr/local/etc/quagga/Quagga.conf
```

O resultado do comando é representado pela Figura 1.52.



```
CORE: n4Router (console)
root@n4Router:/tmp/pycore.58550/n4Router.conf# cat /usr/local/etc/quagga/Quagga.conf
interface eth0
no ipv6 nd suppress-ra
ipv6 nd ra-interval 5
ipv6 nd prefix 2001:db8::/64
no ipv6 nd managed-config-flag
ipv6 nd other-config-flag
ipv6 address 2001:db8::def/64
!
root@n4Router:/tmp/pycore.58550/n4Router.conf# █
```

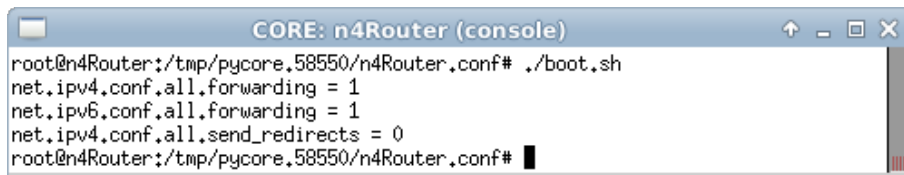
Figura 1.52: *verificação do arquivo de configuração do Quagga após sua edição.*

5. Em paralelo, efetue:

- (a) A coleta dos pacotes trafegados na interface eth0 de n1Host. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
- (b) Efetue os seguintes passos enquanto a coleta é realizada:
- i. Abra um terminal de n4Router e utilize o seguinte comando para iniciar o Quagga com as novas configurações:

```
# ./boot.sh
```

Este comando reinicializa os serviços associados ao roteador, incluindo o serviço de roteamento Quagga. Isto é realizado para que o roteador envie a mensagem RA. O resultado do comando é representado pela Figura 1.53.



```

CORE: n4Router (console)
root@n4Router:~/tmp/pycore.58550/n4Router.conf# ./boot.sh
net.ipv4.conf.all.forwarding = 1
net.ipv6.conf.all.forwarding = 1
net.ipv4.conf.all.send_redirects = 0
root@n4Router:~/tmp/pycore.58550/n4Router.conf#

```

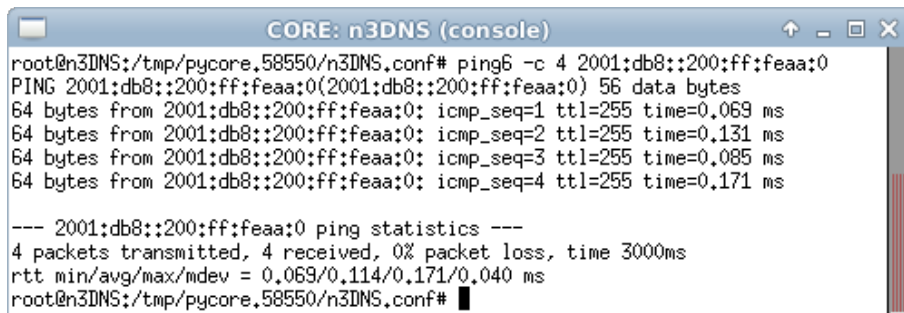
Figura 1.53: resultado esperado da inicialização do Quagga com a nova configuração.

Após reinicializar o serviço Quagga, verifique a configuração de endereços IPv6 nos nós n1Host e n4Router e a conectividade entre eles, conforme descrito no Apêndice C.

- ii. Abra um terminal de n3DNS com um duplo-clique e verifique a conectividade IPv6 com cliente:

```
# ping6 -c 4 2001:db8::200:ff:feaa:0
```

O resultado do comando é representado pela Figura 1.54.



```

CORE: n3DNS (console)
root@n3DNS:~/tmp/pycore.58550/n3DNS.conf# ping6 -c 4 2001:db8::200:ff:feaa:0
PING 2001:db8::200:ff:feaa:0(2001:db8::200:ff:feaa:0) 56 data bytes
64 bytes from 2001:db8::200:ff:feaa:0: icmp_seq=1 ttl=255 time=0.069 ms
64 bytes from 2001:db8::200:ff:feaa:0: icmp_seq=2 ttl=255 time=0.131 ms
64 bytes from 2001:db8::200:ff:feaa:0: icmp_seq=3 ttl=255 time=0.085 ms
64 bytes from 2001:db8::200:ff:feaa:0: icmp_seq=4 ttl=255 time=0.171 ms

--- 2001:db8::200:ff:feaa:0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.069/0.114/0.171/0.040 ms
root@n3DNS:~/tmp/pycore.58550/n3DNS.conf#

```

Figura 1.54: verificação da conectividade entre n3DNS e n1Host.

O endereço IPv6 de n1Host deve ser o IP adquirido por meio do RA, mostrado no passo 5.

- iii. Abra um terminal de n1Host com um duplo-clique e teste o serviço DNS. Para isto utilize o comando:

```
# dig -x 2001:db8::
```

O resultado do comando é representado pela Figura 1.55.



```
CORE: n1Host (console)
root@n1Host:/tmp/pycore.58550/n1Host.conf# dig -x 2001:db8::

;<><> DiG 9.8.1-P1 <>> -x 2001:db8::
;; global options: +cmd
;; connection timed out; no servers could be reached
root@n1Host:/tmp/pycore.58550/n1Host.conf#
```

Figura 1.55: resultado esperado da consulta DNS por n1Host.

Conforme visto anteriormente, o pacote RA enviado pelo roteador por meio do Quagga, não possui a informação relativa ao servidor DNS recursivo a ser utilizado. Neste experimento, essa função será realizada pelo DHCPv6 em modo *stateless*.

- iv. Abra um terminal de n2DHCPv6 e crie os arquivos de configuração do DHCP:

```
# touch dhcpd.conf
# touch dhcpd.leases
```

O resultado dos comandos é representado pela Figura 1.56.



```
CORE: n2DHCPv6 (console)
root@n2DHCPv6:/tmp/pycore.58550/n2DHCPv6.conf# touch dhcpd.conf
root@n2DHCPv6:/tmp/pycore.58550/n2DHCPv6.conf# touch dhcpd.leases
root@n2DHCPv6:/tmp/pycore.58550/n2DHCPv6.conf#
```

Figura 1.56: resultado da criação dos arquivos de configuração do DHCP.

- v. Ainda no terminal de n2DHCPv6, edite o arquivo de configuração do DHCP, localizado em dhcpd.conf:

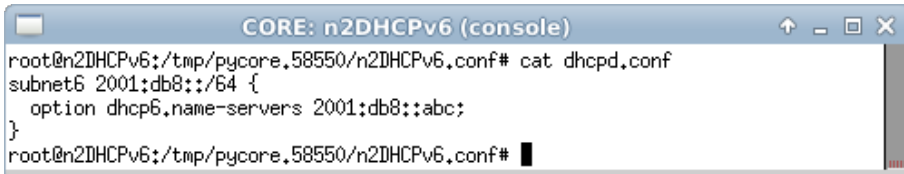
```
subnet6 2001:db8::/64 {
    option dhcp6.name-servers 2001:db8::abc;
}
```

O campo name-servers contém o endereço da máquina que funcionará como servidor DNS. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano.

- vi. Ainda no terminal de n2DHCPv6, verifique o conteúdo do arquivo de configuração do DHCP:

```
# cat dhcpd.conf
```

O resultado do comando é representado pela Figura 1.57.



```

CORE: n2DHCPv6 (console)
root@n2DHCPv6:/tmp/pycore.58550/n2DHCPv6.conf# cat dhcpd.conf
subnet6 2001:db8::/64 {
  option dhcp6.name-servers 2001:db8::abc;
}
root@n2DHCPv6:/tmp/pycore.58550/n2DHCPv6.conf#

```

Figura 1.57: *verificação do arquivo de configuração do DHCP após sua edição.*

- vii. Ainda no terminal de n2DHCPv6, inicie o serviço DHCPv6:

```
# dhcpd -6 -cf dhcpd.conf -lf dhcpd.leases
```

O resultado do comando é representado pela Figura 1.58.



```

CORE: n2DHCPv6 (console)
root@n2DHCPv6:/tmp/pycore.58550/n2DHCPv6.conf# dhcpd -6 -cf dhcpd.conf -lf dhcpd
.leases
Internet Systems Consortium DHCP Server 4.2.4-P2
Copyright 2004-2012 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Wrote 0 leases to leases file.
Bound to *:547
Listening on Socket/5/eth0/2001:db8::/64
Sending on Socket/5/eth0/2001:db8::/64
root@n2DHCPv6:/tmp/pycore.58550/n2DHCPv6.conf#

```

Figura 1.58: *resultado da inicialização do serviço DHCPv6.*

- viii. Abra um terminal de n1Host com um duplo-clique e crie o arquivo de configuração do WIDE-DHCPv6 *client*. Para isto utilize o comando:

```
# touch dhcp6c.conf
```

O resultado do comando é representado pela Figura 1.59.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.58550/n1Host.conf# touch dhcp6c.conf
root@n1Host:/tmp/pycore.58550/n1Host.conf#

```

Figura 1.59: *resultado da criação do arquivo de configuração de WIDE-DHCPv6 client.*

- ix. Ainda no terminal de n1Host, edite o arquivo de configuração do WIDE-DHCPv6 *client*, localizado em dhcp6c.conf:

```
interface eth0 {
    information-only;
    request domain-name-servers;
    script "/etc/wide-dhcpv6/dhcp6c-script";
};
```

A diretiva `information-only` indica que o uso do DHCPv6 será em modo *stateless*, isto é, só irá requerer informações adicionais sem atribuir novos endereços IPv6, enquanto que a diretiva `request domain-name-servers` indica que o cliente requer a informação de qual servidor recursivo de DNS deve ser utilizado. A diretiva `script` aponta para um arquivo de *script* utilizado para atualizar o arquivo `/etc/resolv.conf`, responsável por indicar os servidores DNS utilizados por `n1Host`. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

- x. Ainda no terminal de `n1Host`, verifique o conteúdo do arquivo de configuração do WIDE-DHCPv6 *client*. Utilize o seguinte comando:

```
# cat dhcp6c.conf
```

O resultado do comando é representado pela Figura 1.60.



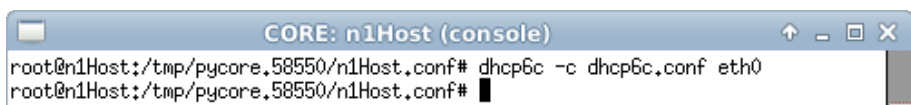
The screenshot shows a terminal window titled "CORE: n1Host (console)". The prompt is "root@n1Host:/tmp/pycore.58550/n1Host.conf#". The user has entered the command "cat dhcp6c.conf". The output of the command is the configuration for the interface eth0, which is identical to the code block shown above. The terminal ends with a prompt "root@n1Host:/tmp/pycore.58550/n1Host.conf#".

Figura 1.60: *verificação do arquivo de configuração do WIDE-DHCPv6 client após sua edição.*

- xi. Ainda no terminal de `n1Host`, inicie o serviço WIDE-DHCPv6 *client*:

```
# dhcp6c -c dhcp6c.conf eth0
```

O resultado do comando é representado pela Figura 1.61.



The screenshot shows a terminal window titled "CORE: n1Host (console)". The prompt is "root@n1Host:/tmp/pycore.58550/n1Host.conf#". The user has entered the command "dhcp6c -c dhcp6c.conf eth0". The terminal shows the command being executed and then returns to the prompt "root@n1Host:/tmp/pycore.58550/n1Host.conf#".

Figura 1.61: *resultado da inicialização do serviço WIDE-DHCPv6 client.*

- xii. Ainda no terminal de `n1Host`, visualize a configuração de DNS. Utilize o comando:

```
# cat /etc/resolv.conf
```

O resultado do comando é representado pela Figura 1.62.

A terminal window titled 'CORE: n1Host (console)' showing the execution of 'cat /etc/resolv.conf'. The output is 'nameserver 2001:db8::abc'.

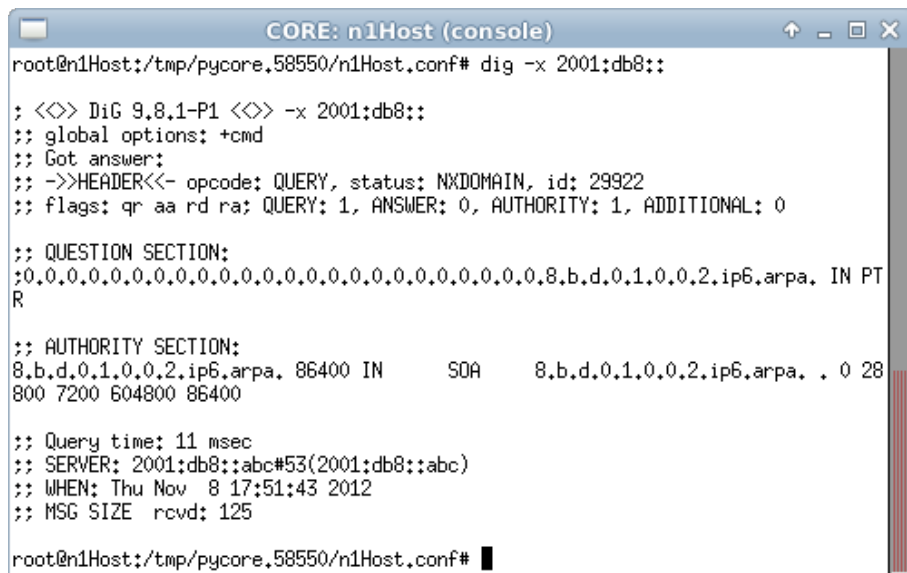
```
root@n1Host:/tmp/pycore.58550/n1Host.conf# cat /etc/resolv.conf
nameserver 2001:db8::abc
root@n1Host:/tmp/pycore.58550/n1Host.conf# █
```

Figura 1.62: resultado da aquisição do endereço IPv6 do DNS recebido por meio do DHCPv6.

xiii. Ainda no terminal de n1Host, verifique o funcionamento do serviço DNS com o comando:

```
# dig -x 2001:db8::
```

O resultado do comando é representado pela Figura 1.63.

A terminal window titled 'CORE: n1Host (console)' showing the execution of 'dig -x 2001:db8::'. The output shows a DNS NXDOMAIN error.

```
root@n1Host:/tmp/pycore.58550/n1Host.conf# dig -x 2001:db8::

;<<> DiG 9.8.1-P1 <<> -x 2001:db8::
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 29922
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. IN PT
R

;; AUTHORITY SECTION:
8.b.d.0.1.0.0.2.ip6.arpa. 86400 IN      SOA     8.b.d.0.1.0.0.2.ip6.arpa. . 0 28
800 7200 604800 86400

;; Query time: 11 msec
;; SERVER: 2001:db8::abc#53(2001:db8::abc)
;; WHEN: Thu Nov  8 17:51:43 2012
;; MSG SIZE rcwd: 125

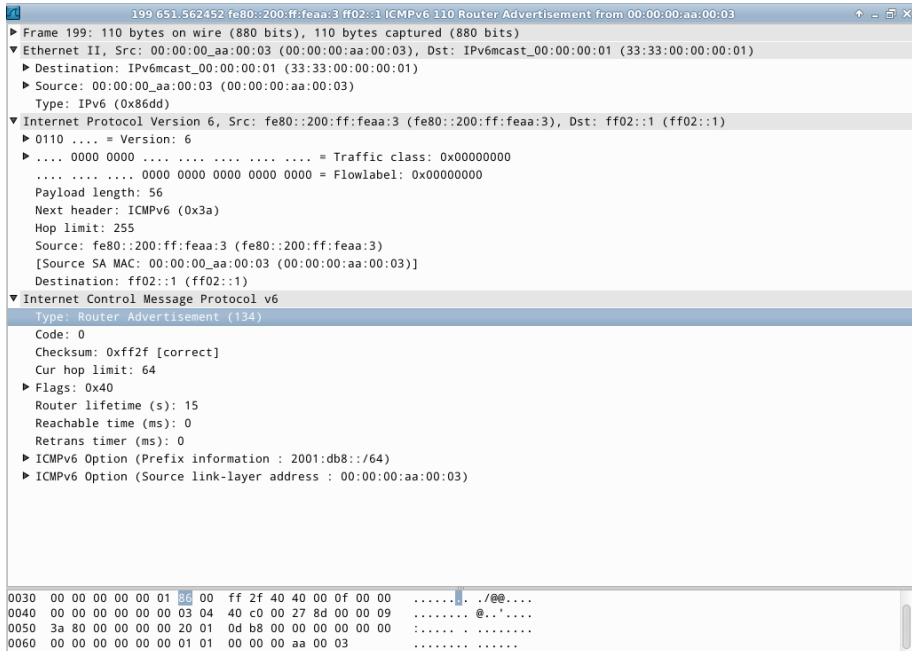
root@n1Host:/tmp/pycore.58550/n1Host.conf# █
```

Figura 1.63: resultado da verificação do serviço DNS em n1Host.

Note que o servidor DNS respondeu à requisição do cliente.

6. Efetue a análise dos pacotes coletados. Aplique o filtro `icmpv6` no Wireshark e procure pelos pacotes RA. Analise os pacotes RA que possuam a opção *Prefix Information* e veja se os dados contidos nos pacotes conferem com a teoria.

Campos importantes do pacote RA, representado pela Figura 1.64:



```

199 651.562452 fe80::200:ff:feaa:3 ff02::1 ICMPv6 110 Router Advertisement from 00:00:00:aa:00:03
  Frame 199: 110 bytes on wire (880 bits), 110 bytes captured (880 bits)
  Ethernet II, Src: 00:00:00_aa:00:03 (00:00:00:aa:00:03), Dst: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
    Destination: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
    Source: 00:00:00_aa:00:03 (00:00:00:aa:00:03)
    Type: IPv6 (0x86dd)
  Internet Protocol Version 6, Src: fe80::200:ff:feaa:3 (fe80::200:ff:feaa:3), Dst: ff02::1 (ff02::1)
    0110 .... = Version: 6
    .... 0000 0000 .... = Traffic class: 0x00000000
    .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 56
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source: fe80::200:ff:feaa:3 (fe80::200:ff:feaa:3)
    [Source SA MAC: 00:00:00_aa:00:03 (00:00:00:aa:00:03)]
    Destination: ff02::1 (ff02::1)
  Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0xff2f [correct]
    Cur hop limit: 64
    Flags: 0x40
    Router lifetime (s): 15
    Reachable time (ms): 0
    Retrans timer (ms): 0
    ICMPv6 Option (Prefix information : 2001:db8::/64)
    ICMPv6 Option (Source link-layer address : 00:00:00:aa:00:03)

```

Figura 1.64: pacote RA mostrado no Wireshark.

Destination (camada Ethernet)

O destino é o endereço (33:33:00:00:00:01), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:00:00:01 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do roteador que enviou a mensagem (00:00:00:aa:00:03).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se a uma mensagem ICMPv6.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface que originou a mensagem, sendo neste caso o roteador (fe80::200:ff:feaa:3).

Destination (camada IPv6)

O destino é o endereço *multicast all-nodes* (ff02::1).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 134 (*Router Advertisement*).

ICMPv6 option (camada ICMPv6)

Indica as opções do pacote ICMPv6:

- *Prefix Information*

Type

Contém o valor 3, que identifica o campo *Prefix Information*.

Autonomous Address-Configuration Flag (A)

Contém o valor 1, indicando que o prefixo deve ser utilizado para autoconfiguração *stateless*.

Preferred lifetime

Marca o tempo, em segundos, em que o endereço é preferencial, isto é, tempo permitido para o uso indistinto do endereço. O valor 0xffffffff indica infinito.

Valid lifetime

Marca o tempo, em segundos, de expiração do endereço gerado. O valor 0xffffffff indica infinito.

Prefix

Contém o prefixo de rede a ser utilizado (2001:db8::).

Prefix length

Contém o tamanho do prefixo da rede (64).

- Source link-layer address

Type

Indica o tipo de opção. Neste caso, *Source link-layer address*.

Link-layer address

Indica o endereço MAC da interface a partir da qual a mensagem de *Router Advertisement* foi enviada, sendo neste caso 00:00:00:aa:00:03.

Aplique o filtro dhcpv6 no Wireshark e procure pelos pacotes *Information-Request* e *Reply*. Analise-os e veja se os dados contidos nos pacotes conferem com o que foi passado na teoria.

Campos importantes do pacote *Information-Request*, representado pela Figura 1.65:

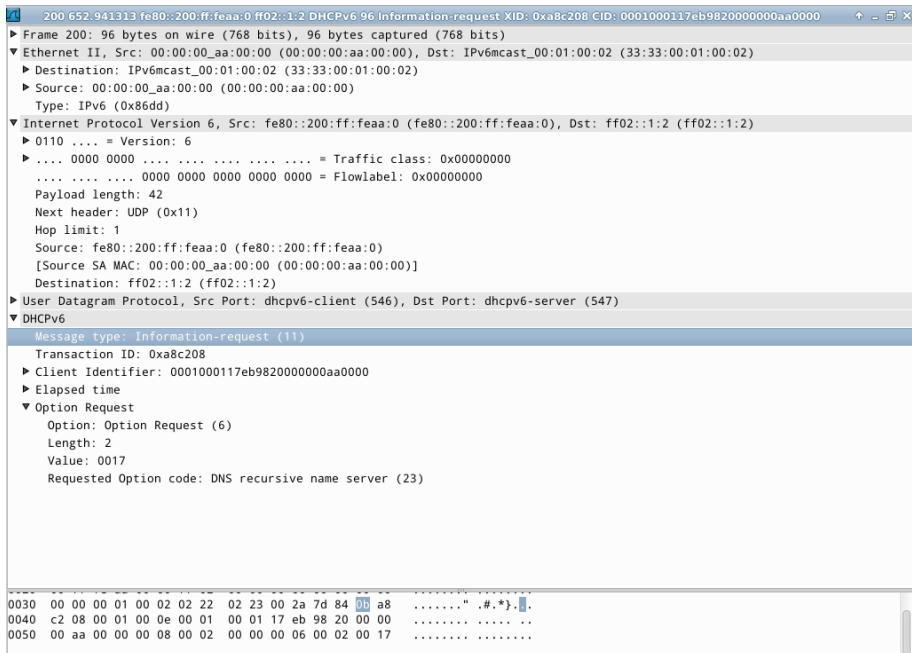


Figura 1.65: pacote *Information-Request* mostrado no Wireshark.

Destination (camada Ethernet)

O destino é o endereço (33:33:00:01:00:02), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:01:00:02 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da máquina solicitante (00:00:00:aa:00:00).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface do dispositivo que enviou a mensagem, ou seja, do cliente (fe80::200:ff:feaa:0).

Destination (camada IPv6)

O destino é o endereço *multicast all-dhcp-agents* (ff02::1:2).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Message type (camada DHCPv6)

Indica por meio do valor 11 que o tipo de mensagem é *Information-Request*.

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Option request (camada DHCPv6)

Indica as opções do pacote DHCPv6:

Requested option code

Indica a solicitação ao servidor DHCP do DNS *Recursive Name Server*, por meio do valor 23.

Campos importantes do pacote *Reply*, representado pela Figura 1.66:

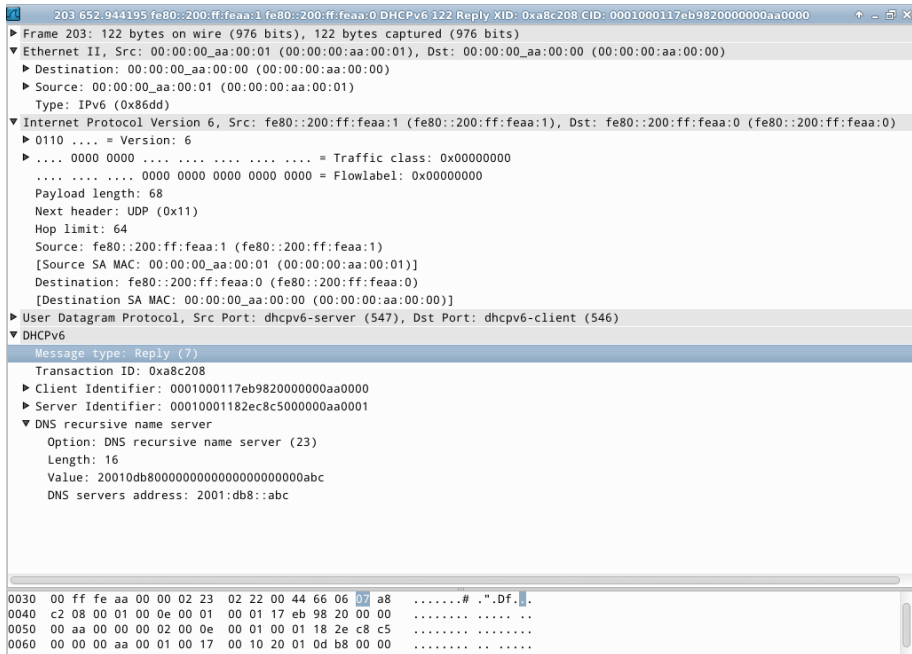


Figura 1.66: pacote *Reply* mostrado no Wireshark.

Destination (camada Ethernet)

O destino é o endereço MAC da máquina solicitante (00:00:00:aa:00:00).

Source (camada Ethernet)

A origem é o endereço MAC da interface do servidor DHCPv6 que enviou a resposta (00:00:00:aa:00:01).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface do dispositivo que enviou a mensagem, ou seja, do servidor DHCPv6 (fe80::200:ff:feaa:1).

Destination (camada IPv6)

O destino é o endereço IPv6 *unicast* de *link-local* do cliente (fe80::200:ff:feaa:0).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Message type (camada DHCPv6)

Indica por meio do valor 7 que o tipo de mensagem é *Reply*.

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Server identifier (camada DHCPv6)

Contém dados da identificação única do servidor baseada no endereço físico.

DNS recursive name server (camada DHCPv6)

Indica as informações relacionadas aos servidores DNS recursivos.

DNS servers address

Indica o endereço IPv6 do servidor DNS requisitado (2001:db8::abc).

7. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 1.9. DHCPv6 *Prefix Delegation*

Objetivo

Esta experiência apresenta o funcionamento da opção *Prefix Delegation* do DHCPv6. O cenário utilizado representa a rede de um provedor de acesso à Internet atribuindo um prefixo de rede /56 IPv6 ao CPE de um cliente. Este cliente, internamente, tem duas sub-redes independentes. Para cada uma deve ser atribuído um prefixo diferente. Após o recebimento do prefixo, o CPE irá dividi-lo em prefixos /64 e os distribuirá em cada uma das sub-redes. Em cada uma delas, o prefixo /64 será anunciado por mensagens RA para que os nós gerem os endereços de suas interfaces utilizando SLAAC.

Para a realização do presente exercício, será utilizada a topologia descrita no arquivo: **1-09-DHCPv6-PD.imn**.

Introdução teórica

O DHCPv6 possui uma opção que é específica desta versão do protocolo, o *Prefix Delegation*. Sua função é informar prefixos de rede a roteadores solicitantes. De forma resumida, seu funcionamento se inicia com o envio de uma requisição de prefixo por um roteador ao servidor DHCPv6 da rede. No servidor DHCPv6, há uma lista pré-configurada de prefixos a serem delegados. O tamanho destes deve ser configurado de acordo com a necessidade de cada rede. Após receber o prefixo solicitado, o roteador fica encarregado de dividi-lo e redistribuí-lo por suas interfaces. Os novos prefixos possuirão o comprimento de 64 *bits* para que possam ser anunciados aos nós.

A entrega de prefixos pelo DHCPv6 atua no modo *stateful*, pois a informação de cada prefixo entregue é registrada. Seu funcionamento é similar ao da atribuição de endereços aos nós.

O *Prefix Delegation* foi criado para suprir uma necessidade das redes IPv6 que não existe no protocolo antigo. Um cenário comum na Internet IPv4 é a atribuição de um único endereço IP a clientes domésticos e de redes maiores a clientes corporativos. Com IPv6, os provedores de acesso

à Internet devem entregar grandes blocos de IPs a qualquer tipo de cliente e não apenas um único endereço. Por exemplo, prefixos /48 a clientes corporativos e prefixos /56 a clientes domésticos. Neste cenário, o servidor DHCPv6, com o conjunto de prefixos a serem entregues configurado, encontra-se na rede do provedor e o roteador solicitante é o CPE do usuário.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-09-DHCPv6-PD.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 1.67, deve aparecer.

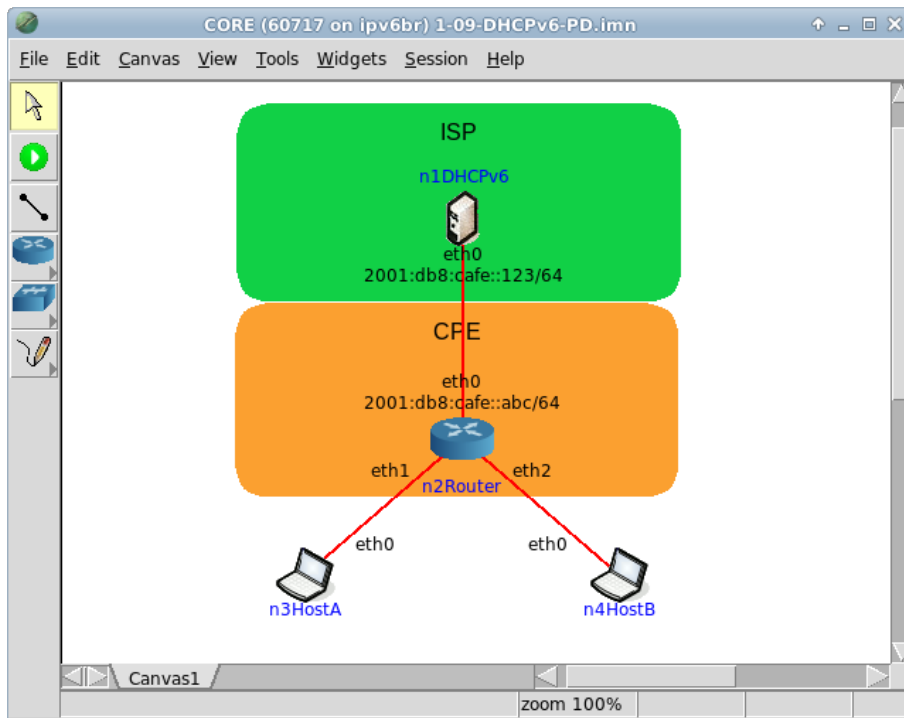


Figura 1.67: topologia da *Experiência 1.9* no CORE.

O objetivo dessa topologia de rede é representar o mínimo necessário para que a troca de mensagens DHCPv6 seja verificada.

2. Configure o DHCP no servidor para enviar o prefixo ao roteador.

- (a) Abra um terminal de n1DHCp6 com um duplo-clique e crie os arquivos de configuração do DHCP:

```
# touch dhcpd.conf
# touch dhcpd.leases
```

O resultado dos comandos é representado pela Figura 1.68.

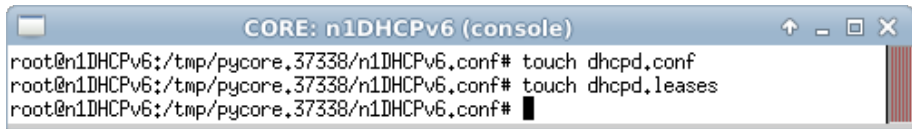


Figura 1.68: resultado da criação dos arquivos de configuração do DHCP.

- (b) Edite o arquivo de configuração do DHCP, localizado em dhcpd.conf:

```
subnet6 2001:db8:cafe::/64 {
    prefix6 2001:db8:cafe:100:: 2001:db8:cafe:f00:: /56;
}
```

O campo `prefix6` contém uma faixa de prefixos de endereços IPv6 que devem ser distribuídos entre os roteadores requisitantes, com os respectivos tamanhos de prefixo. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

- (c) Verifique o conteúdo do arquivo de configuração do DHCP. Utilize o seguinte comando:

```
# cat dhcpd.conf
```

O resultado do comando é representado pela Figura 1.69.

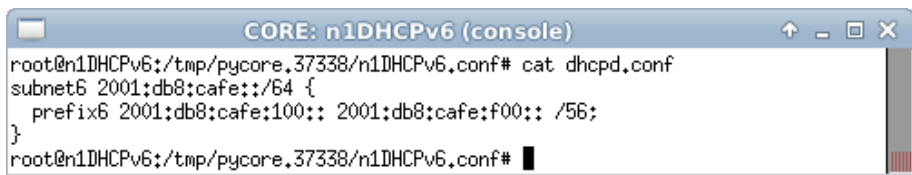


Figura 1.69: verificação do arquivo de configuração do DHCP após sua edição.

- (d) Inicie o serviço DHCPv6:

```
# dhcpd -6 -cf dhcpd.conf -lf dhcpd.leases
```



```

CORE: n1DHCpv6 (console)
root@n1DHCpv6:/tmp/pycore.37338/n1DHCpv6.conf# dhcpd -f -cf dhcpd.conf -lf dhcpd
leases
Internet Systems Consortium DHCP Server 4.2.4-P2
Copyright 2004-2012 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Wrote 0 leases to leases file.
Bound to *:547
Listening on Socket/5/eth0/2001:db8:cafe::64
Sending on Socket/5/eth0/2001:db8:cafe::64
root@n1DHCpv6:/tmp/pycore.37338/n1DHCpv6.conf# █

```

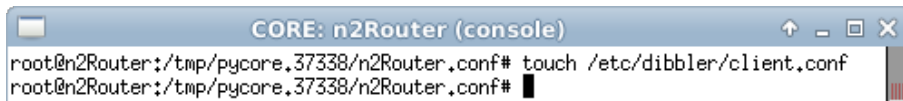
Figura 1.70: resultado da inicialização do serviço DHCPv6.

O resultado do comando é representado pela Figura 1.70.

3. Configure o dibbler-client no roteador para receber as configurações do servidor DHCPv6.
 - (a) Abra um terminal de n2Router e crie os arquivos de configuração do dibbler-client. Para isto digite o seguinte comando:

```
# touch /etc/dibbler/client.conf
```

O resultado do comando é representado pela Figura 1.71.



```

CORE: n2Router (console)
root@n2Router:/tmp/pycore.37338/n2Router.conf# touch /etc/dibbler/client.conf
root@n2Router:/tmp/pycore.37338/n2Router.conf# █

```

Figura 1.71: resultado da criação do arquivo de configuração de dibbler-client.

- (b) Edite o arquivo de configuração do dibbler-client, localizado em /etc/dibbler/client.conf:

```

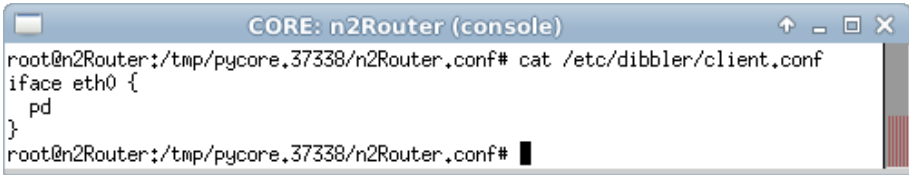
iface eth0 {
    pd
}

```

A diretiva `pd` instrui o cliente a requisitar o serviço de delegação de prefixos ao servidor, alcançado por meio da interface `eth0`. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

- (c) Verifique o conteúdo do arquivo de configuração do dibbler-client:

```
# cat /etc/dibbler/client.conf
```



```

CORE: n2Router (console)
root@n2Router:/tmp/pycore.37338/n2Router.conf# cat /etc/dibbler/client.conf
iface eth0 {
    pd
}
root@n2Router:/tmp/pycore.37338/n2Router.conf# █

```

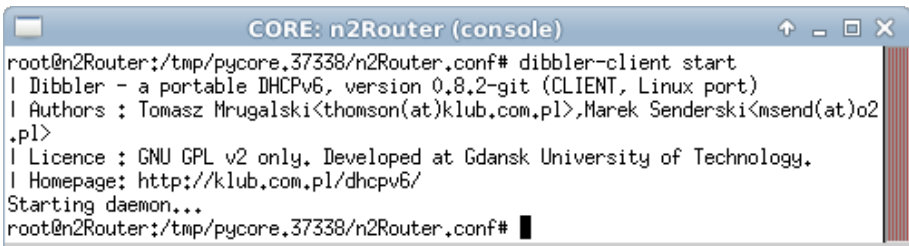
Figura 1.72: *verificação do arquivo de configuração do dibbler-client após sua edição.*

O resultado do comando é representado pela Figura 1.72.

4. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface eth0 de n2Router. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - (b) Efetue os seguintes passos enquanto a coleta é realizada:
 - i. Abra um terminal do roteador com um duplo-clique e inicie o serviço dibbler-client:

```
# dibbler-client start
```

O resultado do comando é representado pela Figura 1.73.



```

CORE: n2Router (console)
root@n2Router:/tmp/pycore.37338/n2Router.conf# dibbler-client start
| Dibbler - a portable DHCPv6, version 0.8.2-git (CLIENT, Linux port)
| Authors : Tomasz Mrugalski<thomson(at)klub.com.pl>,Marek Senderski<msend(at)o2.pl>
| Licence : GNU GPL v2 only. Developed at Gdansk University of Technology.
| Homepage: http://klub.com.pl/dhcpv6/
Starting daemon...
root@n2Router:/tmp/pycore.37338/n2Router.conf# █

```

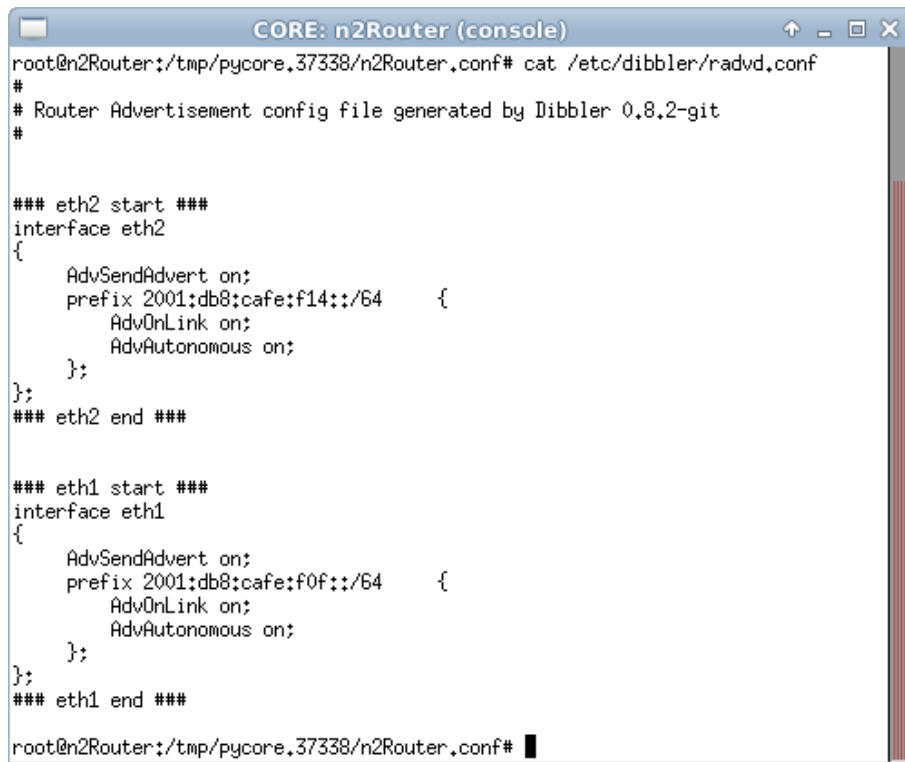
Figura 1.73: *resultado da inicialização do serviço dibbler-client.*

- ii. Espere em torno de 10 segundos.

Dentro do tempo esperado, o roteador recebeu um prefixo /56 para administrar. Nos próximos passos, o roteador será configurado para que divida o prefixo recebido de modo a distribuir prefixos /64 em cada interface compartilhada com um cliente.
- iii. Ainda no terminal do roteador, observe o conteúdo do arquivo de configuração do radvd gerado automaticamente pelo dibbler-client. Para isto utilize o comando:

```
# cat /etc/dibbler/radvd.conf
```

O resultado do comando será similar ao representado pela Figura 1.74.

A screenshot of a terminal window titled "CORE: n2Router (console)". The terminal shows the command "cat /etc/dibbler/radvd.conf" being executed. The output is a configuration file for radvd, generated by Dibbler 0.8.2-git. It contains two interface configurations: eth2 and eth1. Each interface configuration includes settings for AdvSendAdvert, AdvOnLink, and AdvAutonomous, along with a specific IPv6 prefix. The terminal prompt is "root@n2Router:/tmp/pycore.37338/n2Router.conf#".

```
root@n2Router:/tmp/pycore.37338/n2Router.conf# cat /etc/dibbler/radvd.conf
#
# Router Advertisement config file generated by Dibbler 0.8.2-git
#

### eth2 start ###
interface eth2
{
    AdvSendAdvert on;
    prefix 2001:db8:cafe:f14::/64    {
        AdvOnLink on;
        AdvAutonomous on;
    };
};
### eth2 end ###

### eth1 start ###
interface eth1
{
    AdvSendAdvert on;
    prefix 2001:db8:cafe:f0f::/64    {
        AdvOnLink on;
        AdvAutonomous on;
    };
};
### eth1 end ###

root@n2Router:/tmp/pycore.37338/n2Router.conf#
```

Figura 1.74: *visualização do arquivo de configuração do radvd gerado pelo dibbler-client.*

5. Na sequência, efetue paralelamente:
 - (a) A coleta dos pacotes trafegados na interface eth0 de n3HostA. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - (b) Efetue os seguintes passos enquanto a coleta é realizada:
 - i. Abra um terminal de n2Router e valide o arquivo de configuração do serviço radvd gerado pelo dibbler-client:

```
# radvd -c -C /etc/dibbler/radvd.conf
```

O resultado do comando é representado pela Figura 1.75.



```

CORE: n2Router (console)
root@n2Router:/tmp/pycore.37338/n2Router.conf# radvd -c -C /etc/dibbler/radvd.conf
Syntax OK
root@n2Router:/tmp/pycore.37338/n2Router.conf# █
  
```

Figura 1.75: resultado da validação do arquivo de configuração do serviço radvd.

- ii. Ainda no terminal de n2Router, inicie o serviço radvd. Utilize o arquivo de configuração gerado pelo dibbler-client, com o comando:

```
# radvd -C /etc/dibbler/radvd.conf
```

O resultado do comando é representado pela Figura 1.76.



```

CORE: n2Router (console)
root@n2Router:/tmp/pycore.37338/n2Router.conf# radvd -C /etc/dibbler/radvd.conf
root@n2Router:/tmp/pycore.37338/n2Router.conf# █
  
```

Figura 1.76: resultado da inicialização do serviço radvd.

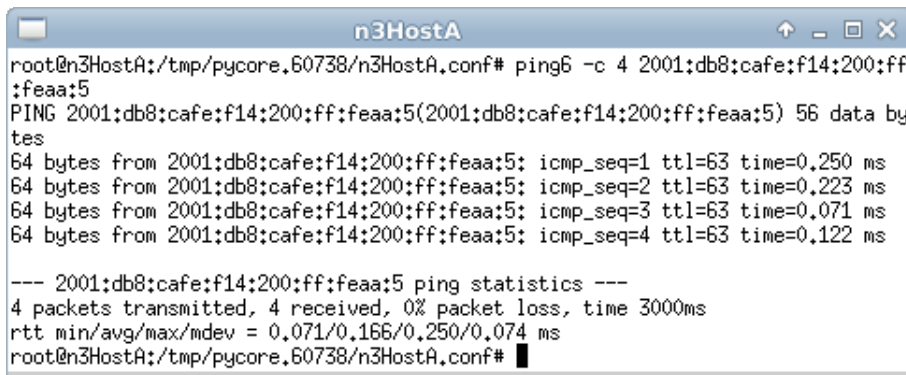
- iii. Espere alguns segundos, p. ex. 10s.
Dentro do tempo esperado, os nós n3HostA e n4HostB receberam os prefixos anunciados pelo roteador.
- iv. Verifique a atribuição de endereço IPv6 de escopo global nos nós n3HostA e n4HostB e as rotas adquiridas pelos mesmos. As instruções de verificação de endereços atribuídos e as rotas configuradas se encontram no Apêndice C.

Note que n3HostA gerou um endereço IPv6 de escopo global baseado em um prefixo /64, enquanto que n4HostB gerou um endereço IPv6 de escopo global baseado em outro prefixo /64. Tais atribuições podem ser verificadas com o arquivo de configuração verificado no passo 4, uma vez que n3HostA está diretamente conectado à interface eth1 do roteador e n4HostB está diretamente conectado à interface eth2 do roteador.

- v. Verifique a atribuição de endereço IPv6 no roteador e suas rotas configuradas. As instruções de verificação de endereços atribuídos e as rotas configuradas se encontram no Apêndice C.
- vi. Abra um terminal de n3HostA e verifique a conectividade IPv6 com n4HostB:

```
# ping6 -c 4 [endereço-n4HostB]
```

O resultado do comando é representado pela Figura 1.77.



```
n3HostA
root@n3HostA:/tmp/pycore.60738/n3HostA.conf# ping6 -c 4 2001:db8:cafe:f14:200:ff:feaa:5
PING 2001:db8:cafe:f14:200:ff:feaa:5(2001:db8:cafe:f14:200:ff:feaa:5) 56 data bytes
64 bytes from 2001:db8:cafe:f14:200:ff:feaa:5: icmp_seq=1 ttl=63 time=0.250 ms
64 bytes from 2001:db8:cafe:f14:200:ff:feaa:5: icmp_seq=2 ttl=63 time=0.223 ms
64 bytes from 2001:db8:cafe:f14:200:ff:feaa:5: icmp_seq=3 ttl=63 time=0.071 ms
64 bytes from 2001:db8:cafe:f14:200:ff:feaa:5: icmp_seq=4 ttl=63 time=0.122 ms

--- 2001:db8:cafe:f14:200:ff:feaa:5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.071/0.166/0.250/0.074 ms
root@n3HostA:/tmp/pycore.60738/n3HostA.conf# █
```

Figura 1.77: verificação de conectividade entre n3HostA e n4HostB.

Observe que o endereço de n4HostB pode variar por se tratar de um mecanismo dinâmico. No exemplo, o endereço IPv6 de n4HostB é 2001:db8:cafe:f14:200:ff:feaa:5.

6. Efetue a análise dos pacotes coletados. Aplique o filtro dhcpv6 no Wireshark e procure pelos pacotes *Solicit*, *Advertise*, *Request* e *Reply*. Analise-os e veja se os dados contidos nos pacotes conferem com o que foi passado na teoria.

Campos importantes do pacote *Solicit*, representado pela Figura 1.78:

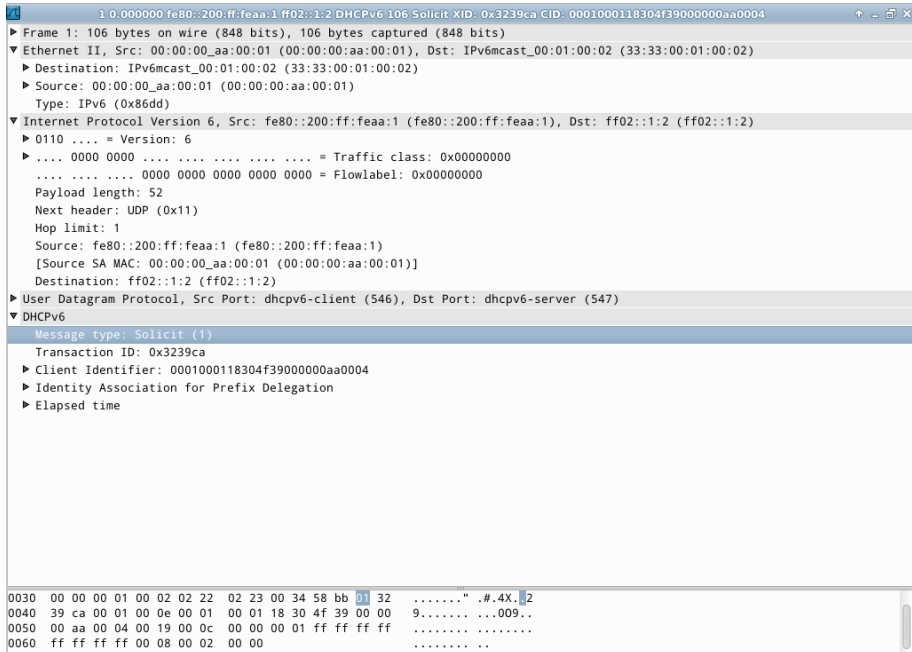


Figura 1.78: pacote *Solicit* mostrado no *Wireshark*.

Destination (camada Ethernet)

O destino é o endereço (33:33:00:01:00:02), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:01:00:02 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do dispositivo que enviou a solicitação (00:00:00:aa:00:01).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface diretamente conectada ao enlace em que se fez a solicitação (fe80::200:ff:feaa:1).

Destination (camada IPv6)

O destino é o endereço *multicast all-dhcp-agents* (ff02::1:2).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Message type (camada DHCPv6)

Indica por meio do valor 1 que o tipo de mensagem é *Solicit*.

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Identity Association for Prefix Delegation (camada DHCPv6)

Identifica a requisição de um prefixo IPv6 para o servidor.

Campos importantes do pacote *Advertise*, representado pela Figura 1.79:

Destination (camada Ethernet)

O destino é o endereço MAC da máquina solicitante (00:00:00:aa:00:01).

Source (camada Ethernet)

A origem é o endereço MAC da interface da máquina que enviou a resposta (00:00:00:aa:00:00).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface do dispositivo que enviou a mensagem, ou seja, do servidor DHCP6 (fe80::200:ff:feaa:0).

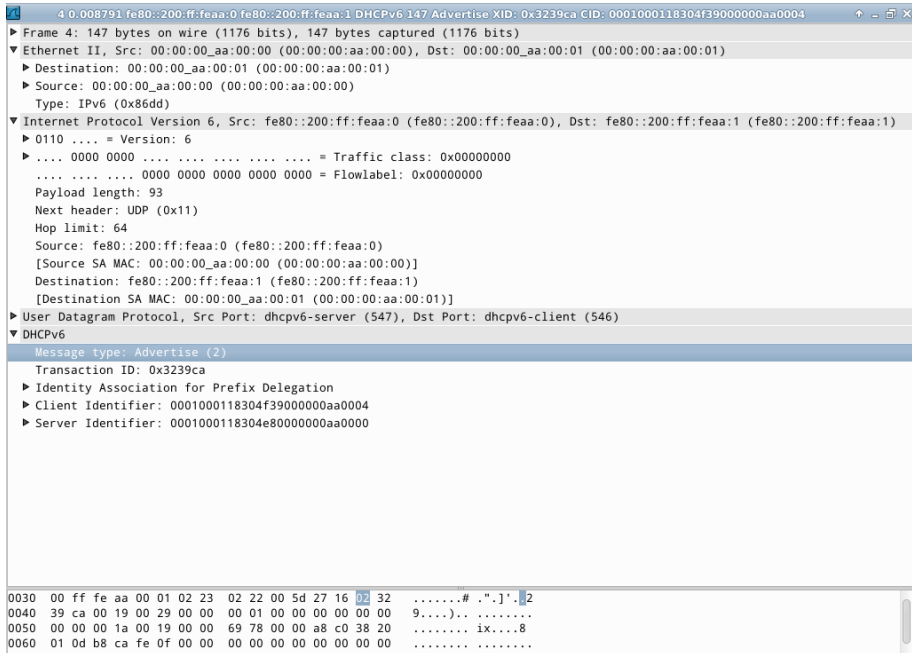


Figura 1.79: pacote Advertise mostrado no Wireshark.

Destination (camada IPv6)

O destino é o endereço *unicast* de *link-local* da máquina solicitante (fe80::200:ff:feaa:1).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Message type (camada DHCPv6)

Indica por meio do valor 2 que o tipo de mensagem é *Advertise*.

Identity Association for Prefix Delegation (camada DHCPv6)

Identifica o uso do prefixo IPv6 entregue para o cliente.

IA Prefix

Contém o prefixo e as configurações que o cliente deve utilizar em sua autoconfiguração, que neste caso é (2001:db8:cafe:f00::/56).

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Server identifier (camada DHCPv6)

Contém dados da identificação única do servidor baseada no endereço físico.

Campos importantes do pacote *Request*, representado pela Figura 1.80:

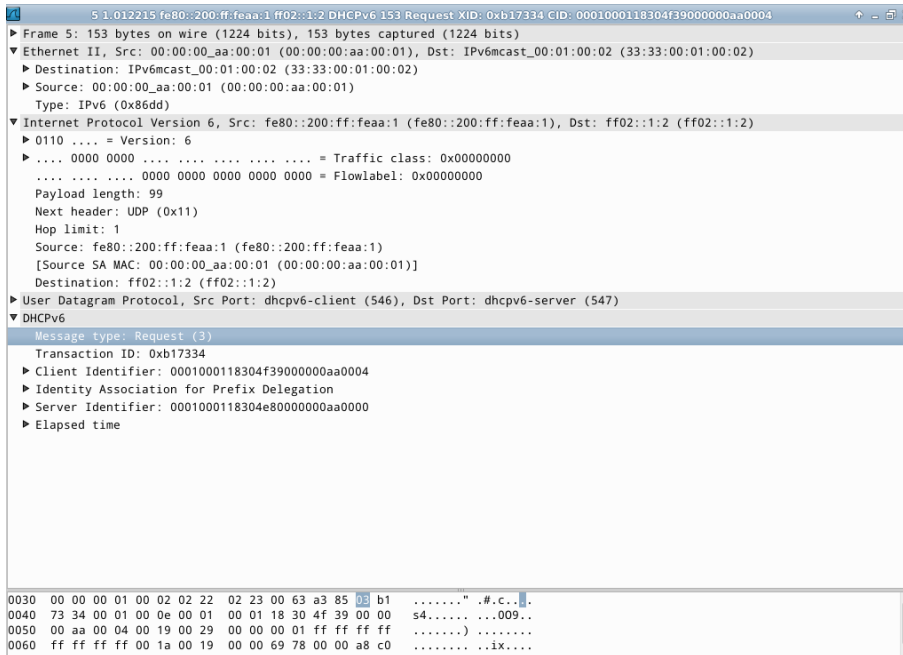


Figura 1.80: pacote *Request* mostrado no *Wireshark*.

Destination (camada Ethernet)

O destino é o endereço (33:33:00:01:00:02), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:01:00:02 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do dispositivo que enviou a requisição (00:00:00:aa:00:01).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface do dispositivo que enviou a mensagem, ou seja, do cliente (fe80::200:ff:feaa:1).

Destination (camada IPv6)

O destino é o endereço *multicast all-dhcp-agents* (ff02::1:2).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Message type (camada DHCPv6)

Indica por meio do valor 3 que o tipo de mensagem é *Request*.

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Identity Association for Prefix Delegation (camada DHCPv6)

Identifica a confirmação do recebimento do prefixo IPv6 para o servidor.

IA Prefix

Contém o prefixo e as configurações que o cliente deve utilizar em sua autoconfiguração, que neste caso é (2001:db8:cafe:f00::/56).

Server identifier (camada DHCPv6)

Contém dados da identificação única do servidor baseada no endereço físico.

Campos importantes do pacote *Reply*, representado pela Figura 1.81:

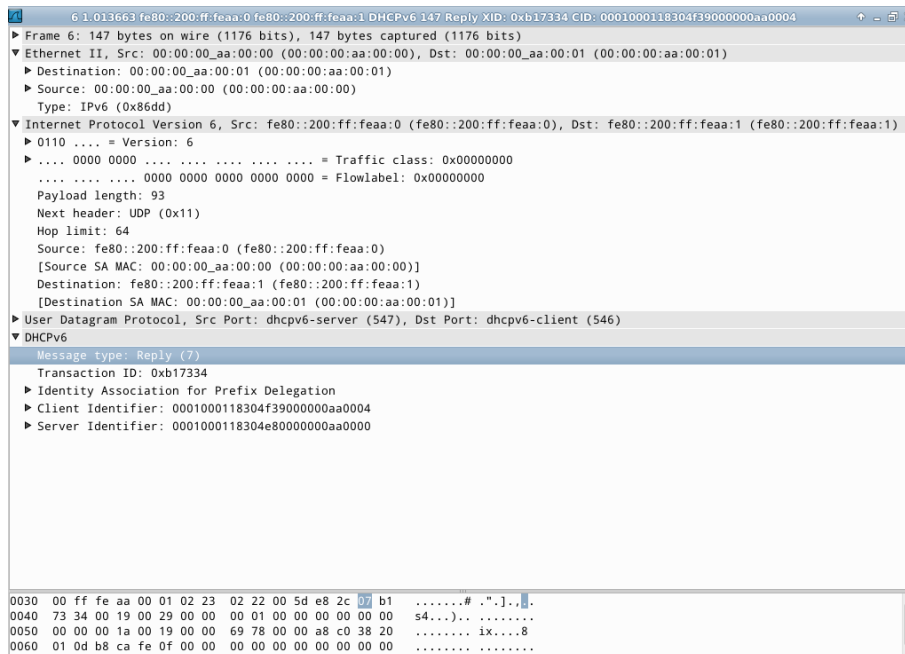


Figura 1.81: pacote Reply mostrado no Wireshark.

Destination (camada Ethernet)

O destino é o endereço MAC da máquina solicitante (00:00:00:aa:00:01).

Source (camada Ethernet)

A origem é o endereço MAC da interface da máquina que enviou a resposta (00:00:00:aa:00:00).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 17 (0x11) refere-se a uma mensagem UDP.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface do dispositivo que enviou a mensagem, ou seja, do servidor DHCPv6 (fe80::200:ff:feaa:0).

Destination (camada IPv6)

O destino é o endereço IPv6 *unicast* de *link-local* do cliente (fe80::200:ff:feaa:1).

Source port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-server. Neste caso o valor é 547.

Destination port (camada UDP)

Indica a porta utilizada pelo serviço dhcpv6-client. Neste caso o valor é 546.

Message type (camada DHCPv6)

Indica por meio do valor 7 que o tipo de mensagem é *Reply*.

Identity Association for Prefix Delegation (camada DHCPv6)

Identifica a confirmação do fornecimento do prefixo IPv6 para o cliente.

IA Prefix

Contém o prefixo e as configurações que o cliente deve utilizar em sua autoconfiguração, que neste caso é (2001:db8:cafe:f00::/56).

Client identifier (camada DHCPv6)

Contém dados da identificação única do cliente baseada no endereço físico.

Server identifier (camada DHCPv6)

Contém dados da identificação única do servidor baseada no endereço físico.

Aplique o filtro `icmpv6` no Wireshark e procure pelos pacotes RA. Analise os pacotes RA que possuam a opção *Prefix Information* e veja se os dados contidos nos pacotes conferem com a teoria.

Campos importantes do pacote RA, representado pela Figura 1.82:

```

1 0.000000 fe80::200:ff:feaa:2 ff02::1 ICMPv6 110 Router Advertisement from 00:00:00:aa:00:02
  Frame 1: 110 bytes on wire (880 bits), 110 bytes captured (880 bits)
  Ethernet II, Src: 00:00:00_aa:00:02 (00:00:00:aa:00:02), Dst: IPv6mcast_00:00:00:01 (33:33:00:00:00:01)
  Internet Protocol Version 6, Src: fe80::200:ff:feaa:2 (fe80::200:ff:feaa:2), Dst: ff02::1 (ff02::1)
    0110 .... = Version: 6
    .... 0000 0000 .... = Traffic class: 0x00000000
    .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 56
    Next header: ICMPv6 (0x3a)
    Hop limit: 255
    Source: fe80::200:ff:feaa:2 (fe80::200:ff:feaa:2)
    [Source SA MAC: 00:00:00_aa:00:02 (00:00:00:aa:00:02)]
    Destination: ff02::1 (ff02::1)
  Internet Control Message Protocol v6
    Type: Router Advertisement (134)
    Code: 0
    Checksum: 0x5c5a [correct]
    Cur hop limit: 64
    Flags: 0x00
    Router lifetime (s): 1800
    Reachable time (ms): 0
    Retrans timer (ms): 0
    ICMPv6 Option (Prefix information : 2001:db8:cafe:f0f::/64)
    ICMPv6 Option (Source link-layer address : 00:00:00:aa:00:02)
0030 00 00 00 00 00 01 3c 00 5c 5a 40 00 07 08 00 00 ..... \Z@.....
0040 00 00 00 00 00 03 04 40 c0 00 01 51 80 00 00 ..... @...Q...
0050 38 40 00 00 00 00 20 01 0d b8 ca fe 0f 0f 00 00 8@.....
0060 00 00 00 00 00 01 01 00 00 00 aa 00 02 .....

```

Figura 1.82: pacote RA mostrado no Wireshark.

Destination (camada Ethernet)

O destino é o endereço (33:33:00:00:00:01), sendo que o prefixo 33:33 indica que a mensagem é um *multicast* na camada Ethernet. O sufixo 00:00:00:01 indica os últimos 32 *bits* do endereço *multicast* IPv6 da mensagem.

Source (camada Ethernet)

A origem é o endereço MAC da interface do roteador que enviou a mensagem (00:00:00:aa:00:02).

Type (camada Ethernet)

Indica que a mensagem utiliza IPv6.

Next header (camada IPv6)

Indica qual é o próximo cabeçalho. Neste caso, o valor 58 (0x3a) refere-se a uma mensagem ICMPv6.

Source (camada IPv6)

A origem é o endereço IP de *link-local* da interface que originou a mensagem, sendo neste caso o roteador (fe80::200:ff:feaa:2).

Destination (camada IPv6)

O destino é o endereço *multicast all-nodes* (ff02::1).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 134 (*Router Advertisement*).

ICMPv6 option (camada ICMPv6)

Indica as opções do pacote ICMPv6:

- *Prefix Information*

Type

Contém o valor 3, que identifica *Prefix Information*.

Autonomous Address-Configuration Flag (A)

Contém o valor 1, indicando que o prefixo deve ser utilizado para autoconfiguração *stateless*.

Preferred lifetime

Marca o tempo, em segundos, em que o endereço é preferencial, isto é, tempo permitido para o uso indistinto do endereço. O valor 0xffffffff indica infinito.

Valid lifetime

Marca o tempo, em segundos, de expiração do endereço gerado. O valor 0xffffffff indica infinito.

Prefix

Contém o prefixo de rede a ser utilizado, que neste caso é (2001:db8:cafe:f0f::).

Prefix length

Contém o tamanho do prefixo da rede (64).

- *Source link-layer address*

Type

Indica o tipo de opção. Neste caso, *Source link-layer address*.

Link-layer address

Indica o endereço MAC da interface a partir da qual a mensagem de *Router Advertisement* foi enviada, sendo neste caso 00:00:00:aa:00:02.

7. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 1.10. *Path MTU Discovery*: mensagem ICMPv6 do tipo *packet too big*

Objetivo

O objetivo desta experiência é mostrar o funcionamento do mecanismo de descoberta de MTU no IPv6. Nela são utilizadas duas redes ligadas por um roteador. Configura-se um valor pequeno para o MTU em uma delas e força-se o envio de pacotes maiores originados na outra. Observa-se então o processo de descoberta de MTU e a fragmentação dos pacotes na origem.

Para isso será utilizada a topologia descrita no arquivo: **1-10-PMTU.imn**.

Introdução teórica

O MTU é o tamanho máximo de pacote suportado em um determinado enlace de rede. Caso seja necessário enviar um pacote maior do que o MTU do enlace é necessário fragmentá-lo.

No IPv6, a fragmentação dos pacotes é realizada apenas na origem. Este procedimento não é realizado (e nem permitido) em roteadores intermediários, como ocorre no protocolo antigo, o IPv4. Isto tem o intuito de reduzir o custo de processamento nos roteadores, seguindo o princípio de manter a inteligência da Internet nas extremidades da rede.

Numa rede IPv6, quem está enviando os pacotes tem então que conhecer o MTU do caminho até o destino. Se o caminho for composto por vários seguimentos com MTUs diferentes, valerá, na prática, o menor deles. Essa informação é obtida por meio do *Path MTU Discovery* (PMTUD), definido na RFC 1981 (McCann *et al.*, 1996).

O processo de PMTUD assume que o MTU de todo o caminho é igual ao do primeiro salto. Caso o tamanho dos pacotes enviados seja maior do que o suportado por algum enlace ao longo do caminho, o roteador irá descartá-lo e enviará uma mensagem ICMPv6 *packet too big*, contendo tanto a mensagem de erro quanto o valor do MTU do enlace seguinte.

Após o recebimento dessa mensagem, o nó de origem passa a limitar o tamanho dos pacotes de acordo com o MTU indicado. Isso é repetido até que o tamanho do pacote seja igual ou inferior ao menor MTU do caminho.

Os dispositivos armazenam o MTU para cada destino em uma tabela chamada *destination cache*, não sendo necessário repetir a descoberta a cada pacote enviado.

Caso o pacote seja enviado a um grupo *multicast*, o tamanho utilizado será o menor MTU de todo o conjunto de destinos.

Implementações minimalistas de IPv6 podem não realizar a descoberta de MTU e utilizar 1280 *bytes* como tamanho máximo para os pacotes.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **1-10-PMTU.imn** localizado no diretório `lab`, dentro do `Desktop`. A topologia de rede, representada pela Figura 1.83, deve aparecer.

O objetivo dessa topologia de rede é representar o mínimo necessário para que o protocolo PMTUD seja percebido.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação e verifique a configuração de endereços IPv6 de todos os nós.

Note que é possível observar dados sobre as interfaces, incluindo o MTU permitido.

3. Altere os valores de MTU nos dispositivos `n1Router` e `n3HostB`.
 - (a) Abra um terminal de `n1Router`, altere o MTU da interface `eth1` e verifique a mudança. Para isso utilize os seguintes comandos:

```
# ip link set eth1 mtu 1400
# ip addr show
```

O resultado dos comandos é representado pela Figura 1.84.

O primeiro comando altera o valor de MTU na interface `eth1`, enquanto o segundo comando é utilizado para visualizar tal mudança.

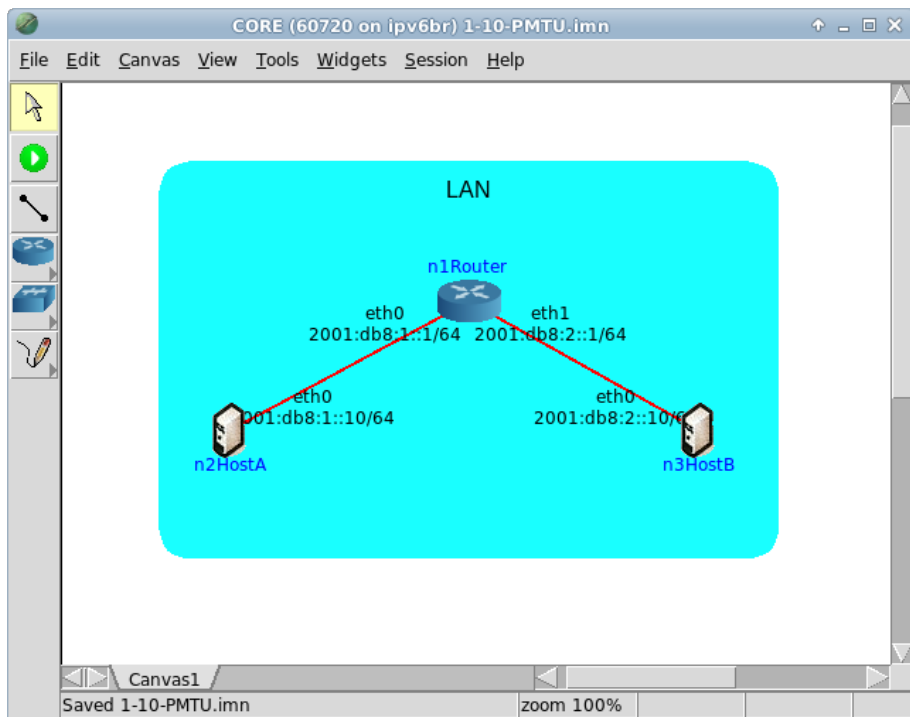


Figura 1.83: topologia da Experiência 1.10 no CORE.

```

CORE: n1Router (console)
root@n1Router:/tmp/pycore.37473/n1Router.conf# ip link set eth1 mtu 1400
root@n1Router:/tmp/pycore.37473/n1Router.conf# ip addr show
94: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
100: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
    link/ether 00:00:00:aa:00:01 brd ff:ff:ff:ff:ff:ff
    inet6 2001:db8:1::1/64 scope global
        valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feaa:1/64 scope link
        valid_lft forever preferred_lft forever
103: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc pfifo_fast state UP
    qlen 1000
    link/ether 00:00:00:aa:00:02 brd ff:ff:ff:ff:ff:ff
    inet6 2001:db8:2::1/64 scope global
        valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feaa:2/64 scope link
        valid_lft forever preferred_lft forever
root@n1Router:/tmp/pycore.37473/n1Router.conf#

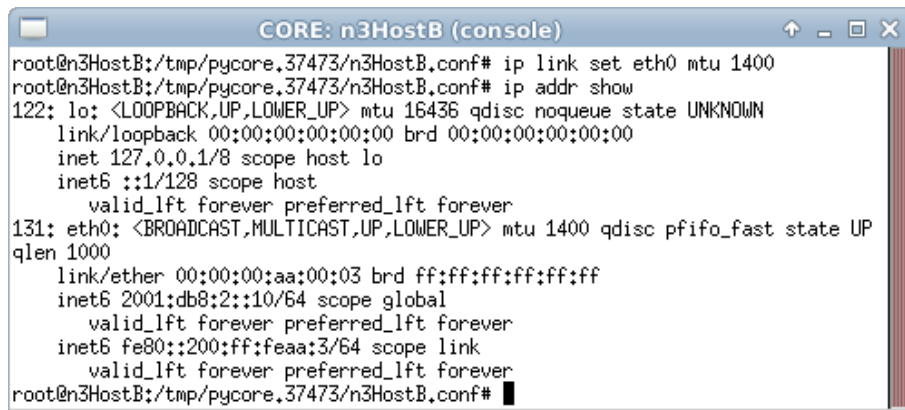
```

Figura 1.84: resultado da alteração do MTU na interface eth1 de n1Router.

- (b) Abra um terminal de n3HostB com um duplo-clique, altere o MTU da interface eth0 e verifique a mudança por meio dos seguintes comandos:

```
# ip link set eth0 mtu 1400
# ip addr show
```

O resultado dos comandos é representado pela Figura 1.85.



```
root@n3HostB:/tmp/pycore.37473/n3HostB.conf# ip link set eth0 mtu 1400
root@n3HostB:/tmp/pycore.37473/n3HostB.conf# ip addr show
122: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
131: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc pfifo_fast state UP
    qlen 1000
    link/ether 00:00:00:aa:00:03 brd ff:ff:ff:ff:ff:ff
    inet6 2001:db8:2::10/64 scope global
        valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feaa:3/64 scope link
        valid_lft forever preferred_lft forever
root@n3HostB:/tmp/pycore.37473/n3HostB.conf# █
```

Figura 1.85: resultado da alteração do MTU na interface eth0 de n3HostB.

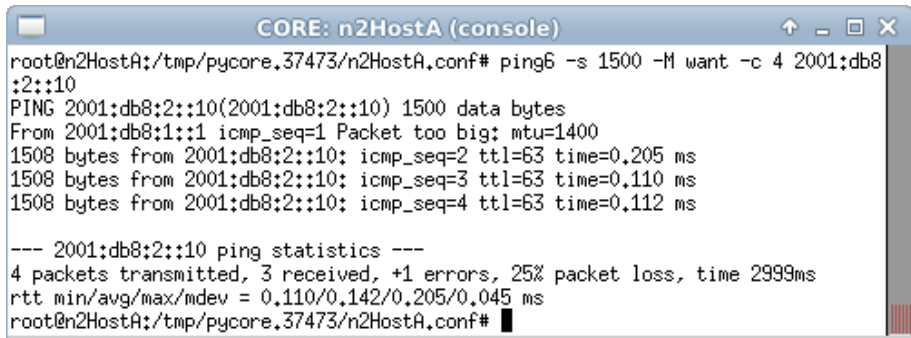
4. Em paralelo, efetue:
- (a) A coleta dos pacotes trafegados na interface eth0 de n2HostA. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
- (b) Abra outro terminal de n2HostA e verifique a conectividade IPv6 com n3HostB. Digite o comando:

```
# ping6 -s 1500 -M want -c 4 2001:db8:2::10
```

O resultado do comando é representado pela Figura 1.86.

Veja que o comando ping6, com os parâmetros apresentados, configura os pacotes enviados para conterem 1500 *bytes* de tamanho, por meio da opção `-s 1500`, e a interface para permitir a fragmentação de pacotes, utilizando a opção `-M want`. O resultado mostra que, para a topologia apresentada, o menor valor de MTU ao longo do caminho foi descoberto

corretamente a partir do primeiro pacote. Este foi descartado ao chegar em uma rede cujo o limite do MTU era de 1400 *bytes*. Na sequência, os pacotes foram enviados fragmentados de acordo com o valor de MTU descoberto e passaram a transitar corretamente pela rede.



```
CORE: n2HostA (console)
root@n2HostA:/tmp/pycore.37473/n2HostA.conf# ping6 -s 1500 -M want -c 4 2001:db8:2::10
PING 2001:db8:2::10(2001:db8:2::10) 1500 data bytes
From 2001:db8:1::1 icmp_seq=1 Packet too big: mtu=1400
1508 bytes from 2001:db8:2::10: icmp_seq=2 ttl=63 time=0.205 ms
1508 bytes from 2001:db8:2::10: icmp_seq=3 ttl=63 time=0.110 ms
1508 bytes from 2001:db8:2::10: icmp_seq=4 ttl=63 time=0.112 ms

--- 2001:db8:2::10 ping statistics ---
4 packets transmitted, 3 received, +1 errors, 25% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.110/0.142/0.205/0.045 ms
root@n2HostA:/tmp/pycore.37473/n2HostA.conf#
```

Figura 1.86: *verificação do PMTUD entre n2HostA e n3HostB.*

5. Efetue a análise dos pacotes capturados. Aplique o filtro `icmpv6` no Wireshark e procure pelos pacotes *packet too big*.

Analise os pacotes *packet too big* e veja se os dados contidos nos pacotes conferem com a teoria.

A partir desta mensagem é possível observar que o pacote não pode ser enviado com o tamanho requisitado e necessita de fragmentação. Logo, tal mensagem informa à origem da comunicação o tamanho máximo que pode ser utilizado para a transmissão de qualquer pacote IPv6 até seu destino.

6. Encerre a simulação, conforme descrito no Apêndice B.

Capítulo 2

Serviços

Experiência 2.1. DNS: consultas DNS

Objetivo

O objetivo deste laboratório é configurar um servidor recursivo para o serviço DNS (*Domain Name System*) em uma rede IPv6 utilizando o *software* BIND.

Será possível observar que as respostas às consultas DNS são independentes do protocolo de rede utilizado, ou seja, um servidor é capaz de responder tanto a consultas AAAA quanto a consultas A, mesmo que possua conexão apenas IPv4 ou apenas IPv6. A resposta deve ser idêntica em ambas as situações.

Para a realização deste exercício será utilizada a topologia descrita no arquivo: **2-01-DNS-recursive.imn**.

Introdução teórica

O *Domain Name System* (DNS) é uma imensa base de dados distribuída em uma estrutura hierárquica, utilizada para a tradução de nomes de domínios em endereços IP e vice-versa.

Os dados associados aos nomes de domínio estão contidos em *Resource Records* ou RRs (Registro de Recursos). Atualmente existe uma grande variedade de tipos de RRs, sendo os mais comuns:

SOA	indica a autoridade sobre uma zona.
NS	indica um servidor de nomes para uma zona.
A	mapeamento de nome para endereço (IPv4).
AAAA	mapeamento de nome para endereço (IPv6).
MX	indica um <i>mail exchanger</i> para um nome (servidor de <i>e-mail</i>).
CNAME	mapeia um nome alternativo (apelido).
PTR	mapeamento de endereço para nome.

O DNS é estruturado da seguinte forma: há os servidores raiz, que têm as informações sobre as principais partes em que esse sistema de nomes é dividido, os domínios de primeiro nível. Há dois tipos de domínios de primeiro nível. Os que usam as siglas que representam os países, chamados de ccTLDs, ou *country code top level domains*: são os nomes que terminam por *.br* para o Brasil, *.pt* para Portugal, etc. Há também os genéricos, que não são vinculados a países específicos, como os *.com*, *.org*, ou *.info*: são chamados de gTLDs ou *generic top level domains*. Há um conjunto de servidores na Internet cuidando de cada um desses domínios de primeiro nível. Eles conhecem todas as informações relevantes sobre eles, por isso são chamados de autoritativos.

Ao se criar um novo nome de domínio qualquer, por exemplo, para um página *Web*, é necessário configurar um servidor em algum lugar na rede, dizendo para qual endereço IP esse nome apontará. Este será também um servidor autoritativo e o servidor do nível hierárquico superior terá as informações dele. Por exemplo, ao criar o autoritativo do domínio *ipv6.br* é preciso informar seu IP ao autoritativo do *.br*.

Os computadores dos usuários utilizam um componente chamado *resolver*, que consulta um servidor chamado de recursivo, geralmente disponibilizado pelo provedor Internet. Os servidores recursivos consultam toda a árvore de servidores autoritativos, começando pela raiz, para encontrar as informações de um determinado nome.

Para que o DNS trabalhe com a versão 6 do protocolo Internet, algumas mudanças foram definidas na RFC 3596 (Thomson *et al.*, 2003).

- Um novo tipo de RR foi criado para armazenar os endereços IPv6 de 128 *bits*, o **AAAA** ou **quad-A**. Sua função é a de traduzir nomes para endereços IPv6, de forma equivalente a do registro do tipo A no IPv4. Caso um dispositivo possua mais de um endereço IPv6, ele deverá ter um registro quad-A para cada um deles. Os registros são representados como se segue:

Exemplo:

```
ipv6.br. IN A      200.160.6.147
ipv6.br. IN AAAA  2001:12ff:0:6172::147
```

- Para resolução de reverso, foi adicionado ao registro **PTR** o domínio ip6.arpa, responsável por traduzir endereços IPv6 em nomes. Em sua representação, o endereço é escrito com o *bit* menos significativo colocado mais a esquerda, de modo que cada dígito hexadecimal seja separado pelo caractere ponto ., como é possível observar no exemplo a seguir:

Exemplo:

```
147.6.160.200.in-addr.arpa                IN PTR ip6.br
7.4.1.0.0.0.0.0.0.0.0.0.0.0.0.2.7.1.6.0.0.0.f.f.2.1.1.0.0.2.ip6.arpa
                                           IN PTR ip6.br
```

Todos os outros tipos de registro DNS não sofreram alterações em sua forma de configuração, apenas foram adaptados para suportar o novo tamanho dos endereços.

É muito importante notar que o protocolo por meio do qual é efetuada uma consulta não influencia na resposta. Ou seja, um servidor é capaz de responder tanto a consultas AAAA, quanto a consultas A, mesmo que possua conexão apenas IPv4 ou apenas IPv6. A resposta deve ser idêntica em ambas as situações.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **2-01-DNS-recursive.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 2.1, deve aparecer.

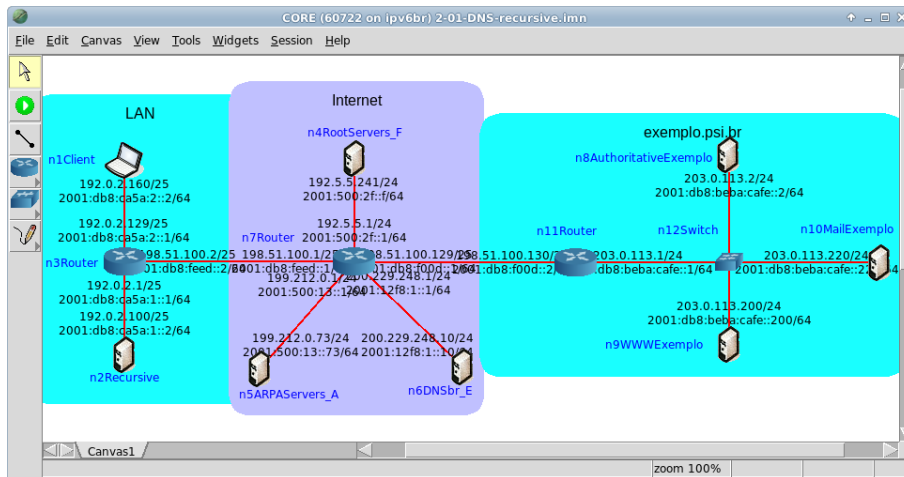


Figura 2.1: topologia da *Experiência 2.1* no CORE.

Nesta topologia há, localizados externamente à rede que configuraremos, a representação de um servidor n9WWWExemplo, que responde pelo nome de domínio `www.exemplo.psi.br` e de um servidor DNS autoritativo n8AuthoritativeExemplo, que possui autoridade sobre este domínio. Além disso, na topologia também se encontram os servidores DNS autoritativos com nomes n4RootServers_F e n6DNSbr_E, os quais são autoridades pelos domínios `.` e `br`, respectivamente.

Na rede local, encontra-se o servidor DNS recursivo n2Recursive, que atuará como um sistema de *cache* relativo às requisições de resolução de nomes aos clientes na LAN, aqui representados por uma máquina denominada n1Client. Desse modo, n2Recursive efetuará as consultas de modo recursivo, consultando os servidores autoritativos, até obter os registros relacionados ao domínio `www.exemplo.psi.br`.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós n1Client, n2Recursive, n4RootServers_F, n5ARPA_Servers_A, n6DNSbr_E, n8AuthoritativeExemplo, n9WWWExemplo e n10MailExemplo e a conectividade entre eles.
3. Neste laboratório, as configurações serão realizadas apenas nos equipamentos do ISP e elas serão iniciadas pelo n2Recursive.

Abra um terminal de n2Recursive com um duplo-clique e visualize o arquivo de configuração do BIND, localizado em `/etc/bind/named.conf`:

```
# cat /etc/bind/named.conf
```

O arquivo de configuração `named.conf` deverá conter as linhas:

```
options {
    allow-query { any; };
    allow-query-cache {
        127.0.0.0/8;
        192.0.2.0/24;
    };
    allow-recursion {
        127.0.0.0/8;
        192.0.2.0/24;
    };
    disable-empty-zone "2.0.192.in-addr.arpa";
    disable-empty-zone "100.51.198.in-addr.arpa";
    disable-empty-zone "113.0.203.in-addr.arpa";
    disable-empty-zone "8.b.d.0.1.0.0.2.ip6.arpa";
};

zone "." {
    type hint;
    file "/etc/bind/db.root";
};
```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano. O arquivo `named.conf` contém as configurações mínimas para o funcionamento de um servidor DNS recursivo. Siga a descrição das opções utilizadas no arquivo de configuração:

`allow-query-cache`

Indica a faixa de endereços IP que tem permissão para realizar consultas ao servidor.

`allow-recursion`

Indica a faixa de endereços IP que tem permissão para realizar consultas recursivas por meio do servidor.

`disable-empty-zone`

Indica que o BIND deve ignorar seu comportamento padrão para os prefixos designados para testes TEST-NET-1, TEST-NET-2 e TEST-NET-3 para IPv4, definidos na RFC 5737 (Arkko *et al.*, 2010) e para o prefixo de documentação para IPv6, definido na RFC 3849 (Huston *et al.*, 2004).

`zone “.”`

Indica como devem ser configuradas as consultas relativas à zona raiz, sendo nesse caso uma zona não gerenciada pelo servidor, fato que pode ser verificado pela opção `type hint` e as informações de acesso estão armazenadas no arquivo `/etc/bind/db.root`.

Com estas configurações, o servidor recursivo responderá apenas a requisições recebidas em conexões IPv4.

4. O BIND possui ferramentas que verificam a sintaxe dos arquivos de configuração que podem auxiliar na resolução de problemas relacionados ao funcionamento do DNS. Deste modo, antes de iniciar o processo do BIND, verifique se o arquivo de configuração está correto:

```
# named-checkconf -p /etc/bind/named.conf
```

O resultado do comando é representado pela Figura 2.2.



```

CORE: n2Recursive (console)
root@n2Recursive:/tmp/pycore.37442/n2Recursive.conf# named-checkconf -p /etc/bi
nd/named.conf
options {
    allow-query-cache {
        127.0.0.0/8;
        192.0.2.0/24;
    };
    allow-recursion {
        127.0.0.0/8;
        192.0.2.0/24;
    };
    disable-empty-zone "2.0.192.in-addr.arpa";
    disable-empty-zone "100.51.198.in-addr.arpa";
    disable-empty-zone "113.0.203.in-addr.arpa";
    disable-empty-zone "8.b.d.0.1.0.0.2.ip6.arpa";
    allow-query {
        "any";
    };
};
zone "." {
    type hint;
    file "/etc/bind/db.root";
};
root@n2Recursive:/tmp/pycore.37442/n2Recursive.conf#

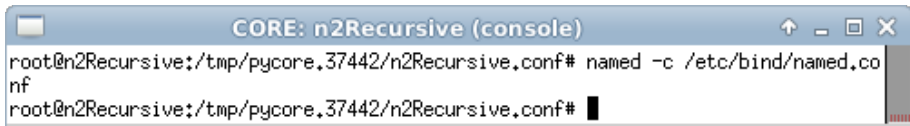
```

Figura 2.2: *verificação do arquivo de configuração do BIND.*

5. Caso o passo anterior não tenha apresentado erros de execução, inicie o processo do BIND:

```
# named -c /etc/bind/named.conf
```

O resultado do comando é representado pela Figura 2.3.



```

CORE: n2Recursive (console)
root@n2Recursive:/tmp/pycore.37442/n2Recursive.conf# named -c /etc/bind/named.co
nf
root@n2Recursive:/tmp/pycore.37442/n2Recursive.conf#

```

Figura 2.3: *resultado da inicialização do serviço DNS BIND.*

6. Abra um terminal de n1Client com um duplo-clique e edite o arquivo localizado em `/etc/resolv.conf`, de modo que a máquina comece a utilizar n2Recursive para a realização de consultas DNS. Substitua o conteúdo existente por:

```
nameserver 192.0.2.100
```

Note que essa regra configura apenas o endereço IPv4 de n2Recursive. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano.

7. Ainda no terminal de n1Client, realize uma consulta DNS ao registro AAAA do domínio `www.exemplo.psi.br`, ou seja, ao endereço IPv6 associado a este domínio. Para isto utilize o seguinte comando:

```
# host -t AAAA www.exemplo.psi.br
```

O resultado do comando é representado pela Figura 2.4.



Figura 2.4: resultado da consulta ao registro AAAA de `www.exemplo.psi.br`.

Observe que mesmo com o n2Recursive configurado para ser acessado apenas por conexões IPv4, o servidor é capaz de responder a requisições de endereços IPv6. Isso demonstra que as informações armazenadas no banco de dados do servidor DNS são independentes da versão do protocolo de rede utilizado na comunicação.

8. Ainda no terminal de n1Client, realize uma nova consulta DNS aos registros do domínio `www.exemplo.psi.br`, desta vez sem o parâmetro `-t AAAA`:

```
# host www.exemplo.psi.br
```

O resultado do comando é representado pela Figura 2.5.

Observe que a resposta contém tanto o endereço IPv4 quanto o endereço IPv6 associados ao domínio pesquisado.

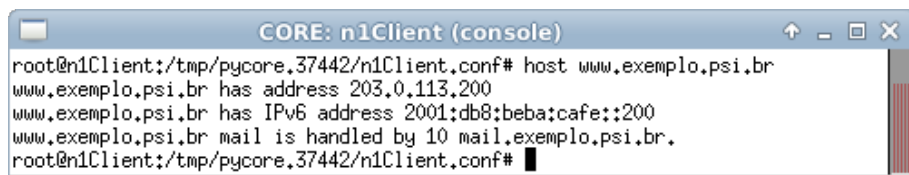


Figura 2.5: resultado da consulta por registros do nome de domínio `www.exemplo.psi.br`.

9. Habilite o servidor DNS recursivo para aceitar requisições vindas por conexões IPv6.
 - (a) Abra um terminal de n2Recursive com um duplo-clique e edite o arquivo de configuração do BIND, localizado em `/etc/bind/named.conf`, de modo a acrescentar os trechos destacados em **negrito**:

```
options {
    allow-query { any; };
    allow-query-cache {
        127.0.0.0/8;
        192.0.2.0/24;
        ::1/128;
        2001:db8:ca5a::/48;
    };
    allow-recursion {
        127.0.0.0/8;
        192.0.2.0/24;
        ::1/128;
        2001:db8:ca5a::/48;
    };
    listen-on-v6 { any; };
    disable-empty-zone "2.0.192.in-addr.arpa";
    disable-empty-zone "100.51.198.in-addr.arpa";
    disable-empty-zone "113.0.203.in-addr.arpa";
    disable-empty-zone "8.b.d.0.1.0.0.2.ip6.arpa";
};

zone "." {
    type hint;
    file "/etc/bind/db.root";
};
```

A opção `listen-on-v6` com o valor `any` permite que qualquer endereço IPv6 disponível em `n2Recursive` receba consultas DNS, enquanto os prefixos adicionados em `allow-query-cache` e `allow-recursion` permitem que os endereços dos prefixos IPv6 especificados realizem consultas. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

- (b) Ainda no terminal de `n2Recursive`, verifique novamente se o arquivo de configuração foi gerado corretamente. Utilize o seguinte comando:

```
# named-checkconf -p /etc/bind/named.conf
```

O resultado do comando é representado pela Figura 2.6.

- (c) Caso o passo anterior não tenha apresentado erros de execução, reinicie o processo do BIND para que a alteração seja aplicada por meio dos comandos:

```
# killall named
# named -c /etc/bind/named.conf
```

O resultado do comando é representado pela Figura 2.7.

- (d) Abra um terminal de `n1Client` com e edite o arquivo localizado em `/etc/resolv.conf`, de modo que a máquina comece a utilizar também o endereço IPv6 de `n2Recursive` para a realização de consultas DNS. Adicione ao conteúdo existente a linha:

```
nameserver 2001:db8:ca5a:1::2
```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.



```

CORE: n2Recursive (console)
root@n2Recursive:/tmp/pycore.37442/n2Recursive.conf# named-checkconf -p /etc/bi
d/named.conf
options {
    listen-on-v6 {
        "any";
    };
    allow-query-cache {
        127.0.0.0/8;
        192.0.2.0/24;
        ::1/128;
        2001:db8:ca5a::/48;
    };
    allow-recursion {
        127.0.0.0/8;
        192.0.2.0/24;
        ::1/128;
        2001:db8:ca5a::/48;
    };
    disable-empty-zone "2.0.192.in-addr.arpa";
    disable-empty-zone "100.51.198.in-addr.arpa";
    disable-empty-zone "113.0.203.in-addr.arpa";
    disable-empty-zone "8.b.d.0.1.0.0.2.ip6.arpa";
    allow-query {
        "any";
    };
};
zone "." {
    type hint;
    file "/etc/bind/db.root";
};
root@n2Recursive:/tmp/pycore.37442/n2Recursive.conf#

```

Figura 2.6: verificação do arquivo de configuração do BIND após sua edição.



```

CORE: n2Recursive (console)
root@n2Recursive:/tmp/pycore.37442/n2Recursive.conf# killall named
root@n2Recursive:/tmp/pycore.37442/n2Recursive.conf# named -c /etc/bind/named.co
nf
root@n2Recursive:/tmp/pycore.37442/n2Recursive.conf#

```

Figura 2.7: resultado da reinicialização do serviço DNS BIND.

- (e) Ainda no terminal de `n1Client`, realize uma consulta DNS ao registro A do domínio `www.exemplo.psi.br`, ou seja, ao endereço IPv4 associado a este domínio. Efetue a consulta forçando que a mesma seja feita por uma conexão IPv6:

```
# host -t A -6 www.exemplo.psi.br
```

O resultado do comando é representado pela Figura 2.8.

A terminal window titled "CORE: n1Client (console)" showing the execution of a command. The prompt is "root@n1Client:/tmp/pycore.37442/n1Client.conf#". The command entered is "host -t A -6 www.exemplo.psi.br". The output is "www.exemplo.psi.br has address 203.0.113.200". The prompt is repeated at the end of the output.

```
CORE: n1Client (console)
root@n1Client:/tmp/pycore.37442/n1Client.conf# host -t A -6 www.exemplo.psi.br
www.exemplo.psi.br has address 203.0.113.200
root@n1Client:/tmp/pycore.37442/n1Client.conf#
```

Figura 2.8: resultado da consulta ao registro A de `www.exemplo.psi.br`.

Assim como ocorreu na verificação do passo 7, mesmo com a realização da requisição por meio de uma conexão IPv6, o servidor DNS recursivo foi capaz de responder por endereços IPv4.

- (f) Faça testes para verificar o funcionamento do servidor DNS. Pode-se utilizar comandos como `dig`, `ping` e `ping6` para verificar a conectividade, resolução de endereço reverso, etc. Alguns exemplos são:

```
# host 2001:db8:beba:cafe::220
# ping www.exemplo.psi.br
# ping6 www.exemplo.psi.br
# nslookup -type=ANY exemplo.psi.br
```

10. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 2.2. DNS: configurando um servidor autoritativo

Objetivo

O objetivo desta experiência é configurar um servidor autoritativo de DNS (*Domain Name System*) preexistente em uma rede com o protocolo antigo, o IPv4, para funcionar com IPv6. Será analisada a configuração inicial do servidor e, em seguida, esta será modificada: (i) fazendo o próprio servidor responder por conexões IPv6, (ii) adicionando registros AAAA para informar os endereços IPv6 correspondentes aos nomes, (iii) configurando o reverso dos endereços IPv6.

Para a realização deste exercício será utilizada a topologia descrita no arquivo: **2-02-DNS-authoritative.imn**.

Introdução teórica

Veja a introdução teórica da Experiência 2.1.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **2-02-DNS-authoritative.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 2.9, deve aparecer.

Nesta topologia há, localizados externamente à rede que configuraremos, a representação de um servidor DNS recursivo n2Recursive responsável por receber requisições de nomes de seus clientes e reencaminhá-las para servidores autoritativos e de um cliente n1Client, que será utilizado para realizar as requisições DNS, testando assim as configurações aplicadas. No ISP, encontram-se dois servidores representando os serviços de *e-mail* e *Web* e um servidor DNS autoritativo n8AuthoritativeExemplo, responsável por responder a requisições ao domínio exemplo.psi.br.

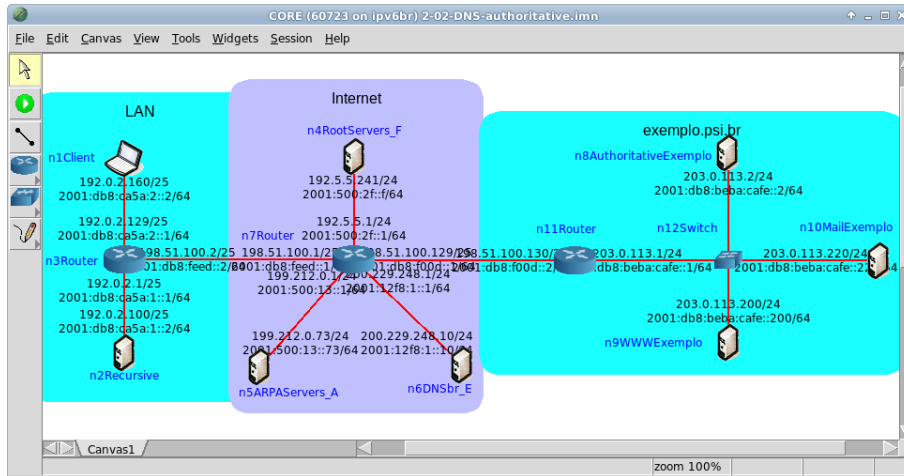


Figura 2.9: topologia da *Experiência 2.2* no CORE.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós n1Client, n2Recursive, n4RootServers_F, n5ARPA_Servers_A, n6DNSbr_E, n8AuthoritativeExemplo, n9WWWExemplo, n10MailExemplo e a conectividade entre eles.
3. Neste laboratório, as configurações serão realizadas apenas nos equipamentos do ISP. Inicialmente, analise os arquivos de configuração do BIND localizados no servidor DNS autoritativo do ISP, o n8AuthoritativeExemplo. Este servidor já está configurado para responder a requisições por registros tipo A e de endereçamento reverso IPv4.
 - (a) Abra um terminal de n8AuthoritativeExemplo com um duplo-clique e visualize o arquivo de configuração do BIND, localizado em /etc/bind/named.conf:

```
# cat /etc/bind/named.conf
```

O arquivo de configuração named.conf deverá conter as linhas:

```
options {
    allow-query { any; };
    listen-on { any; };
    recursion no;
};
zone "exemplo.psi.br" {
    type master;
```

```
file "/etc/bind/exemplo.psi.br.zone";
};
zone "113.0.203.in-addr.arpa" {
    type master;
    file "/etc/bind/203.0.113.db";
};
```

Este arquivo contém as configurações básicas necessárias para o funcionamento do servidor DNS autoritativo. No bloco de comandos `options`, há especificações que controlam o comportamento global do servidor. Neste exemplo, são listadas as seguintes opções:

`allow-query`

Lista quais endereços IP possuem permissão para realizar requisições, sendo neste exemplo aceitas requisições oriundas de qualquer IP.

`listen-on`

Lista os endereços IPv4 e portas habilitados a responderem às requisições DNS. Este exemplo apresenta a configuração padrão, responder em qualquer interface através da porta 53 (IANA, 2014).

`recursion`

Indica se o servidor é capaz (`yes`) ou não (`no`) de reencaminhar requisições a outros servidores autoritativos.

Após as opções, encontra-se a lista de arquivos com as zonas conhecidas pelo servidor e dos arquivos com as respectivas informações.

`exemplo.psi.br`

Indica sobre qual domínio o servidor DNS possui autoridade para responder requisições (`type master`).

`113.0.203.in-addr.arpa`

Indica qual a zona de endereçamento reverso IPv4 o servidor responde.

- (b) Ainda no terminal de `n8AuthoritativeExemplo`, visualize o arquivo da zona `exemplo.psi.br`, localizado em `/etc/bind/exemplo.psi.br.zone`:

```
# cat /etc/bind/exemplo.psi.br.zone
```

O arquivo deverá conter as linhas:

```
exemplo.psi.br. 30 IN SOA ns.exemplo.psi.br. ipv6.nic.br. (
```

```

                2013022890 1800 900 604800 60 )
30 IN NS ns.exemplo.psi.br.
;; DNS server
ns          30 IN A 203.0.113.2
;; WWW server
www        30 IN A 203.0.113.200
;; Mail server
          30 IN MX 10 mail.exemplo.psi.br.
mail       30 IN A 203.0.113.220
;; SPF policy
          30 IN TXT "v=spf1 a mx ip4:203.0.113.220 -all"
          30 IN SPF "v=spf1 a mx ip4:203.0.113.220 -all"
;; SSH server
ssh        30 IN CNAME www.exemplo.psi.br.

```

Este arquivo apresenta os registros e diretivas relacionados à zona exemplo.psi.br. A diretiva \$TTL (*Time To Live*) indica o tempo que os registros devem permanecer no *cache* sem que sejam atualizados, podendo ser expresso em segundos, minutos, horas, dias ou semanas. Apesar da recomendação ser que tal valor corresponda a pelo menos um dia, neste exemplo o TTL está configurado para 30 segundos, apenas para que os exercícios possam ser demonstrados. Este valor é representado pelo número trinta presente em todas as linhas de configuração dos registros.

- (c) Ainda no terminal de n8AuthoritativeExemplo, visualize o arquivo /etc/bind/203.0.113.db. Utilize o seguinte comando:

```
# cat /etc/bind/203.0.113.db
```

O arquivo deverá conter as linhas:

```
113.0.203.in-addr.arpa. 86400 IN SOA ns.exemplo.psi.br. ipv6.nic.br. (
                                2013022890 86400 3600 604800 86400 )
                                86400 IN NS ns.exemplo.psi.br.
;; DNS server
2                                86400 IN PTR ns.exemplo.psi.br.
;; WWW server
200                              86400 IN PTR www.exemplo.psi.br.
;; Mail server
220                              86400 IN PTR mail.exemplo.psi.br.
```

Este arquivo apresenta os registros e diretivas relacionados à zona de endereçamento reverso IPv4.

4. Efetue consultas DNS para testar as configurações do servidor DNS autoritativo da rede ISP.
 - (a) Abra um terminal de n1Client com um duplo-clique e efetue uma consulta sobre `www.exemplo.psi.br`:

```
# host www.exemplo.psi.br
```

O resultado do comando é representado pela Figura 2.10.

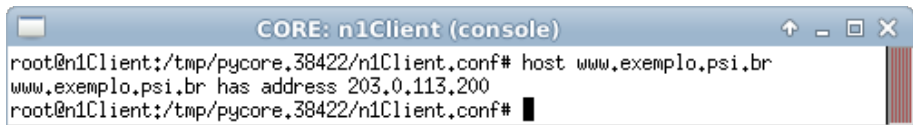


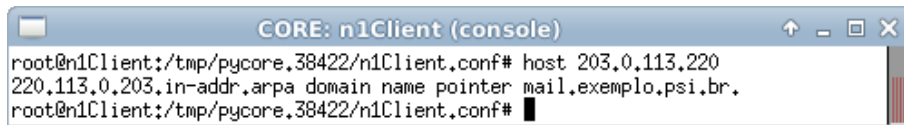
Figura 2.10: *resultado da consulta sobre `www.exemplo.psi.br` oriunda de n1Client.*

Observe a partir da resposta obtida que para o nome consultado há um endereço IPv4 associado, o 203.0.113.200.

- (b) Ainda no terminal de n1Client, pode-se realizar consultas relativas à resolução de endereço reverso. Verifique um exemplo por meio do comando:


```
# host 203.0.113.220
```

O resultado do comando é representado pela Figura 2.11. O conteúdo das respostas é obtido por meio de consultas DNS recursivas. Note que apesar dos servidores do ISP possuírem endereços IPv6 em suas interfaces de rede, nenhuma consulta DNS feita retornou um endereço IPv6 como resposta. Isto ocorreu porque tal informação não consta nos arquivos de zona do servidor DNS autoritativo do ISP.



```

CORE: n1Client (console)
root@n1Client:/tmp/pycore.38422/n1Client.conf# host 203.0.113.220
203.0.113.220.in-addr.arpa domain name pointer mail.exemplo.psi.br.
root@n1Client:/tmp/pycore.38422/n1Client.conf#

```

Figura 2.11: resultado da consulta sobre 203.0.113.220 oriunda de n1Client.

5. A seguir, configure o servidor autoritativo para que ele seja capaz tanto de receber requisições por meio de conexões IPv6, quanto responder endereços IPv6 às consultas realizadas.
 - (a) Abra um terminal de n8AuthoritativeExemplo com um duplo-clique e edite o arquivo de configuração do BIND, localizado em /etc/bind/named.conf, de modo a acrescentar o trecho destacado em **negrito**:

```

options {
    allow-query { any; };
    listen-on { any; };
    listen-on-v6 { any; };
    recursion no;
};

```

A opção `listen-on-v6` com o valor `any` permite que qualquer endereço IPv6 disponível em n8AuthoritativeExemplo receba consultas DNS. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano.

- (b) Ainda no terminal de n8AuthoritativeExemplo, verifique se o arquivo de configuração foi gerado corretamente. Execute o seguinte comando:

```
# named-checkconf -p /etc/bind/named.conf
```

O resultado do comando é representado pela Figura 2.12.



```

CORE: n8AuthoritativeExemplo (console)
root@n8AuthoritativeExemplo:/tmp/pycore,38422/n8AuthoritativeExemplo.conf# named-checkconf -p /etc/bind/named.conf
options {
    listen-on {
        "any";
    };
    listen-on-v6 {
        "any";
    };
    recursion no;
    allow-query {
        "any";
    };
};
zone "exemplo.psi.br" {
    type master;
    file "/etc/bind/exemplo.psi.br.zone";
};
zone "113.0.203.in-addr.arpa" {
    type master;
    file "/etc/bind/203.0.113.db";
};
root@n8AuthoritativeExemplo:/tmp/pycore,38422/n8AuthoritativeExemplo.conf#

```

Figura 2.12: *verificação do arquivo de configuração do BIND após sua edição.*

- (c) Ainda no terminal de `n8AuthoritativeExemplo`, edite o arquivo relacionado à zona `exemplo.psi.br`, localizado em `/etc/bind/exemplo.psi.br.zone` de modo a adicionar os endereços IPv6 aos nomes e registros previamente configurados e alterar os parâmetros das políticas de SPF. Para tal modificação, acrescente os trechos destacados em **negrito**:

```

exemplo.psi.br. 30 IN SOA ns.exemplo.psi.br. ipv6.nic.br. (
                    2013022890 1800 900 604800 60 )
                    30 IN NS  ns.exemplo.psi.br.
;; DNS server
ns                30 IN A  203.0.113.2
ns                30 IN AAAA 2001:db8:beba:cafe::2
;; WWW server
www               30 IN A  203.0.113.200
www               30 IN AAAA 2001:db8:beba:cafe::200
;; Mail server
mail              30 IN MX 10 mail.exemplo.psi.br.
mail              30 IN A  203.0.113.220
mail              30 IN AAAA 2001:db8:beba:cafe::220
;; SPF policy

```

```

30 IN TXT "v=spf1 a mx ip4:203.0.113.220 ip6:2001:db8:beba:cafe::220 -all"
30 IN SPF "v=spf1 a mx ip4:203.0.113.220 ip6:2001:db8:beba:cafe::220 -all"
;; SSH server
ssh          30 IN CNAME www.exemplo.psi.br.

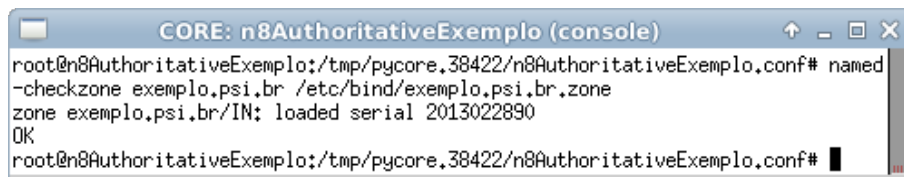
```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano. Além de adicionar os registros AAAA aos nomes de domínios previamente cadastrados, também foram adicionados parâmetros às políticas de SPF.

- (d) Ainda no terminal de `n8AuthoritativeExemplo`, verifique se o arquivo de configuração foi gerado corretamente. Utilize o seguinte comando:

```
# named-checkzone exemplo.psi.br /etc/bind/exemplo.psi.br.zone
```

O resultado do comando é representado pela Figura 2.13.



```

CORE: n8AuthoritativeExemplo (console)
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf# named
-checkzone exemplo.psi.br /etc/bind/exemplo.psi.br.zone
zone exemplo.psi.br/IN: loaded serial 2013022890
OK
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf# █

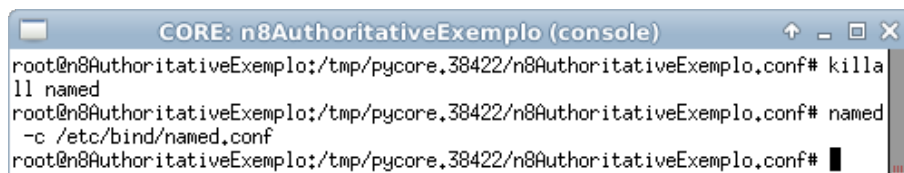
```

Figura 2.13: *verificação do arquivo da zona exemplo.psi.br após sua edição.*

- (e) Caso os passos anteriores não tenham apresentado erros de execução, reinicie o processo do BIND para que a alteração seja aplicada. Para isto, execute os seguintes comandos:

```
# killall named
# named -c /etc/bind/named.conf
```

O resultado do comando é representado pela Figura 2.14.



```

CORE: n8AuthoritativeExemplo (console)
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf# killa
ll named
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf# named
-c /etc/bind/named.conf
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf# █

```

Figura 2.14: *resultado da reinicialização do serviço DNS BIND.*

- (f) Efetue consultas DNS para testar as novas configurações do servidor DNS autoritativo da rede ISP em relação à zona `www.exemplo.psi.br`. Abra um terminal de `n1Client` com um duplo-clique e efetue uma consulta por meio do comando:

```
# host www.exemplo.psi.br
```

O resultado do comando é representado pela Figura 2.15.



Figura 2.15: resultado da consulta sobre `www.exemplo.psi.br` oriunda de `n1Client`.

Além das informações obtidas no passo 4, também existe agora um endereço IPv6 associado ao nome `www.exemplo.psi.br`.

- (g) Ainda no terminal de `n1Client`, repita os testes para os outros endereços e compare os resultados. Para obter uma resposta mais completa, utilize o seguinte comando:

```
# nslookup -type=ANY exemplo.psi.br
```

O resultado do comando é representado pela Figura 2.16.

6. Para finalizar o exercício, configure o servidor autoritativo para responder a requisições de endereçamento reverso IPv6.
- (a) Abra um terminal de `n8AuthoritativeExemplo` com um duplo-clique e visualize o arquivo localizado em `/etc/bind/2001-db8-beba.db`:

```
# cat /etc/bind/2001-db8-beba.db
```

O arquivo deverá conter as linhas:

```
a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa. 86400 IN SOA ns.exemplo.psi.br. ipv6.nic.br. (
                                2013022890 86400 3600 604800 86400 )
                                86400 IN NS ns.exemplo.psi.br.

;; DNS server
```

```

2.0.0.0.0.0.0.0.0.0.0.0.0.0.e.f.a.c 86400 IN PTR ns.exemplo.psi.br.
;; WWW server
;; SSH server
0.0.2.0.0.0.0.0.0.0.0.0.0.0.e.f.a.c 86400 IN PTR www.exemplo.psi.br.
;; Mail server
0.2.2.0.0.0.0.0.0.0.0.0.0.0.e.f.a.c 86400 IN PTR mail.exemplo.psi.br.

```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano.

Este arquivo apresenta os registros e diretivas relacionados à zona de endereçamento reverso IPv6. Ele possui as mesmas funcionalidades e características do arquivo 203.0.113.db analisado no passo 3.



```

CORE: n1Client (console)
root@n1Client:/tmp/pycore.38422/n1Client.conf# nslookup -type=ANY exemplo.psi.br
Server:          192.0.2.100
Address:         192.0.2.100#53

Non-authoritative answer:
exemplo.psi.br
    origin = ns.exemplo.psi.br
    mail addr = ipv6.nic.br
    serial = 2013022890
    refresh = 1800
    retry = 900
    expire = 604800
    minimum = 60
exemplo.psi.br mail exchanger = 10 mail.exemplo.psi.br.
exemplo.psi.br nameserver = ns.exemplo.psi.br.

Authoritative answers can be found from:
exemplo.psi.br nameserver = ns.exemplo.psi.br.
mail.exemplo.psi.br internet address = 203.0.113.220
mail.exemplo.psi.br has AAAA address 2001:db8:beba:cafe::220

root@n1Client:/tmp/pycore.38422/n1Client.conf# █

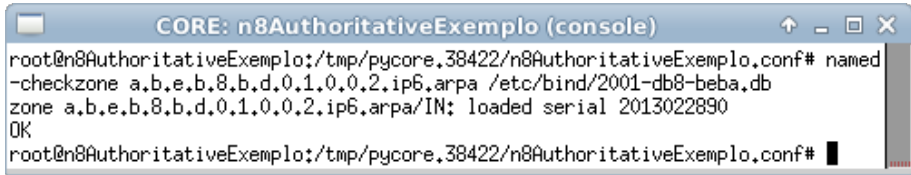
```

Figura 2.16: resultado da consulta mais completa sobre exemplo.psi.br oriunda de n1Client.

- (b) Ainda no terminal de n8AuthoritativeExemplo, verifique se o arquivo de configuração foi gerado corretamente. Para isto utilize o seguinte comando:

```
# named-checkzone a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa /etc/bind/2001-db8-beba.db
```

O resultado do comando é representado pela Figura 2.17.



```

CORE: n8AuthoritativeExemplo (console)
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf# named
-checkzone a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa /etc/bind/2001-db8-beba.db
zone a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa/IN: loaded serial 2013022890
OK
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf#

```

Figura 2.17: *verificação do arquivo da zona a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa após sua edição.*

- (c) Para o uso da informação editada, é necessário cadastrar a zona de endereçamento reverso IPv6 no arquivo de configuração do serviço DNS BIND. Ainda no terminal de `n8AuthoritativeExemplo`, edite o arquivo localizado em `/etc/bind/named.conf`, de modo a acrescentar as seguintes linhas:

```

zone "a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa" {
    type master;
    file "/etc/bind/2001-db8-beba.db";
};

```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

- (d) Ainda no terminal de `n8AuthoritativeExemplo`, verifique se o arquivo de configuração foi gerado corretamente. Utilize o seguinte comando:

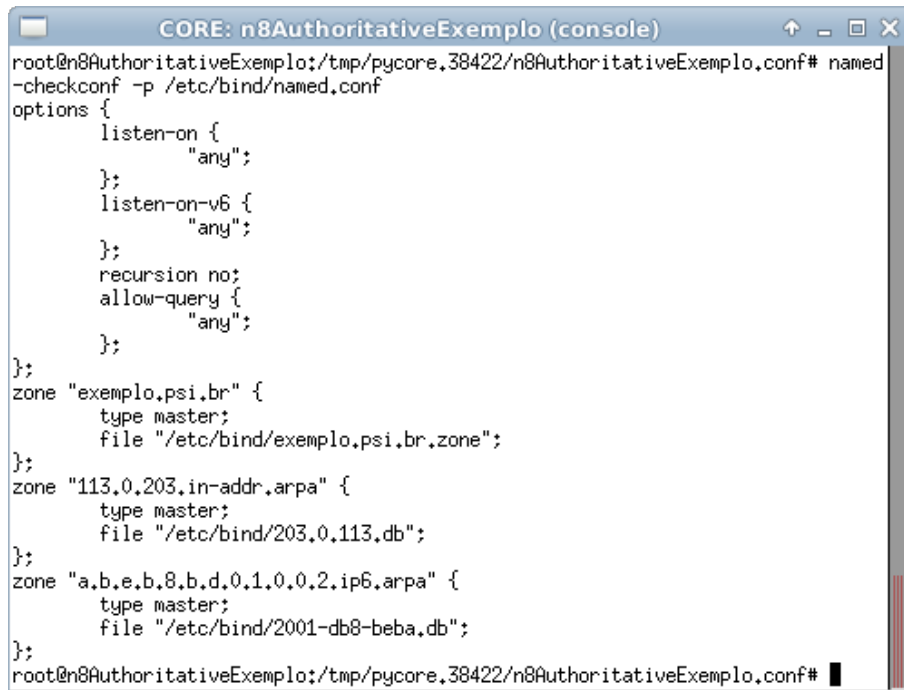
```
# named-checkconf -p /etc/bind/named.conf
```

O resultado do comando é representado pela Figura 2.18.

- (e) Caso os passos anteriores não tenham apresentado erros de execução, reinicie o processo do BIND para que a alteração seja aplicada:

```
# killall named
# named -c /etc/bind/named.conf
```

O resultado do comando é representado pela Figura 2.19.

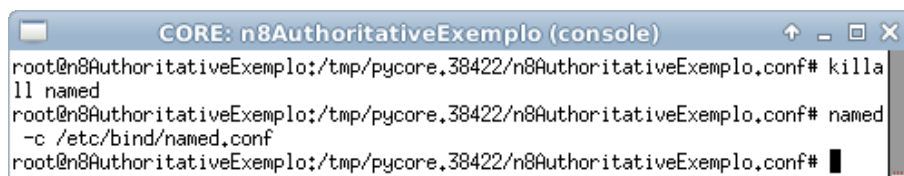


```

CORE: n8AuthoritativeExemplo (console)
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf# named
-checkconf -p /etc/bind/named.conf
options {
    listen-on {
        "any";
    };
    listen-on-v6 {
        "any";
    };
    recursion no;
    allow-query {
        "any";
    };
};
zone "exemplo.psi.br" {
    type master;
    file "/etc/bind/exemplo.psi.br.zone";
};
zone "113.0.203.in-addr.arpa" {
    type master;
    file "/etc/bind/203.0.113.db";
};
zone "a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa" {
    type master;
    file "/etc/bind/2001-db8-beba.db";
};
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf#

```

Figura 2.18: *verificação do arquivo de configuração do BIND após sua edição.*



```

CORE: n8AuthoritativeExemplo (console)
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf# killall
named
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf# named
-c /etc/bind/named.conf
root@n8AuthoritativeExemplo:/tmp/pycore.38422/n8AuthoritativeExemplo.conf#

```

Figura 2.19: *resultado da reinicialização do serviço DNS BIND.*

- (f) Efetue consultas DNS para testar as novas configurações do servidor DNS autoritativo da rede ISP em relação à zona de endereçamento reverso IPv6. Abra um terminal de `niClient` e efetue uma consulta:

```
# host 2001:db8:beba:cafe::220
```

O resultado do comando é representado pela Figura 2.20.



```

CORE: n1Client (console)
root@n1Client:/tmp/pycore.38422/n1Client.conf# host 2001:db8:beba:cafe::220
0.2.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.f.a.c.a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa domain
name pointer mail.exemplo.psi.br.
root@n1Client:/tmp/pycore.38422/n1Client.conf# █

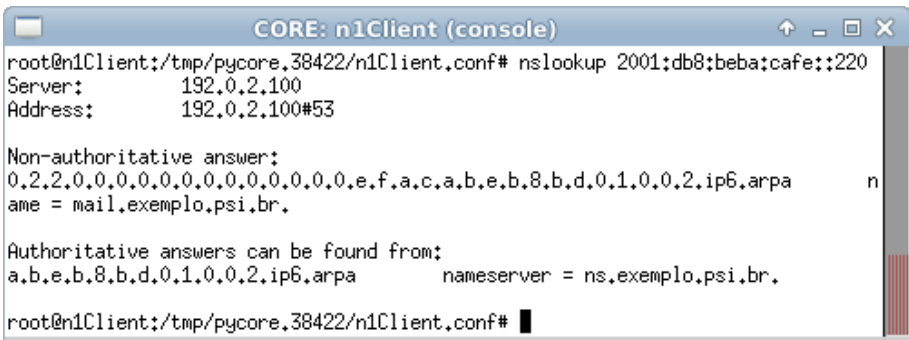
```

Figura 2.20: resultado da consulta sobre `2001:db8:beba:cafe::220` oriunda de `n1Client`.

- (g) Ainda no terminal de `n1Client`, repita os testes para os outros endereços e compare os resultados. Para obter uma resposta mais completa, utilize o seguinte comando:

```
# nslookup 2001:db8:beba:cafe::220
```

O resultado do comando deve contar pelo menos as informações apresentadas na Figura 2.21.



```

CORE: n1Client (console)
root@n1Client:/tmp/pycore.38422/n1Client.conf# nslookup 2001:db8:beba:cafe::220
Server:          192.0.2.100
Address:         192.0.2.100#53

Non-authoritative answer:
0.2.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.f.a.c.a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa      n
ame = mail.exemplo.psi.br.

Authoritative answers can be found from:
a.b.e.b.8.b.d.0.1.0.0.2.ip6.arpa      nameserver = ns.exemplo.psi.br.
root@n1Client:/tmp/pycore.38422/n1Client.conf# █

```

Figura 2.21: resultado da consulta mais completa sobre `2001:db8:beba:cafe::220` oriunda de `n1Client`.

7. É possível realizar uma série de testes de conectividade utilizando o nome de uma máquina. Algumas opções podem ser feitas com a utilização dos comandos `ping` ou `ping6`, `traceroute` ou `traceroute6` e `mtr`. Alguns exemplos são:

```
# ping www.exemplo.psi.br
# ping6 www.exemplo.psi.br
# mtr www.exemplo.psi.br
# traceroute6 mail.exemplo.psi.br
```

8. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 2.3. HTTP: configuração IPv6 no Apache para novas páginas *Web*

Objetivo

O objetivo deste laboratório é demonstrar que o servidor *Web* Apache, por padrão, já apresenta suporte ao IPv6. O serviço Apache será iniciado em um dos nós e, sem a necessidade de qualquer configuração, será possível observar que o servidor responderá a requisições por meio de conexões IPv6.

Para a realização do presente exercício será utilizada a topologia descrita no arquivo: **2-03-HTTP-apache-new.imn**.

Introdução teórica

Servidor *Web* é a aplicação responsável por responder a requisições do protocolo HTTP feita por clientes (navegadores, também chamados de *browsers*). Esta resposta normalmente é uma página *Web*, em formato HTML, que pode conter textos, imagens, vídeos, etc.

Os principais servidores *Web* utilizados hoje em dia já apresentam suporte ao IPv6. O Apache HTTP Server, o mais utilizado na *Web*, suporta por padrão o IPv6 desde sua versão 2.0. Outro servidor *Web* bem conhecido, o Nginx, recebeu suporte ao IPv6 em sua versão 08.22. Ao contrário do que ocorre com o Apache, o suporte ao IPv6 não é habilitado por padrão no Nginx, porém é simples habilitá-lo.

Na maior parte das vezes não é necessário fazer nada para que o servidor *Web* funcione com IPv6 ou basta habilitá-lo em um arquivo de configuração. Nos casos em que o servidor possui configurações complexas, com *VirtualServers* atrelados a endereços IPv4 específicos, é necessário especificar também os endereços IPv6.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **2-03-HTTP-apache-new.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 2.22, deve aparecer.

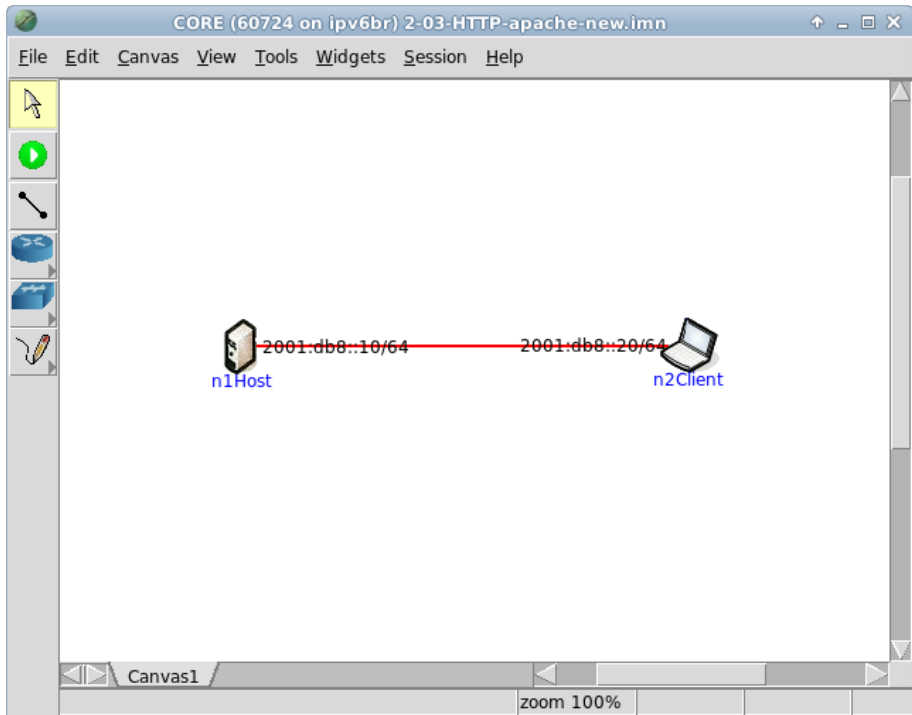


Figura 2.22: topologia da *Experiência 2.3* no CORE.

O objetivo dessa topologia de rede é representar o mínimo necessário para que o serviço HTTP seja analisado.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós n1Host e n2Client e a conectividade entre eles.
3. Abra um terminal de n1Host com um duplo-clique e inicie o serviço HTTP Apache. Para isto digite o seguinte comando:

```
# /etc/init.d/apache2 start
```



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.55814/n1Host.conf# /etc/init.d/apache2 start
* Starting web server apache2
apache2: apr_sockaddr_info_get() failed for n1Host
apache2: Could not reliably determine the server's fully qualified domain name,
using 127.0.0.1 for ServerName
[ OK ]
root@n1Host:/tmp/pycore.55814/n1Host.conf# █

```

Figura 2.23: resultado esperado da inicialização do serviço HTTP Apache em n1Host.

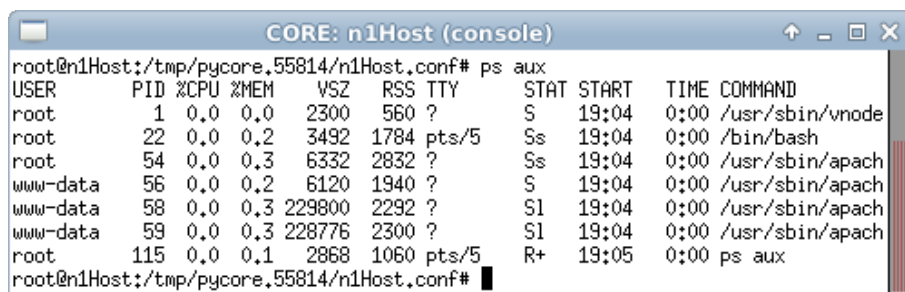
O resultado do comando é representado pela Figura 2.23.

Os avisos mostrados ocorrem pelo fato dos nomes de domínio e do servidor não estarem configurados adequadamente para a máquina. Entretanto, tais mensagens não afetam o funcionamento do serviço para a experiência.

4. Ainda no terminal de n1Host, verifique o funcionamento do serviço Apache:

```
# ps aux
```

O resultado do comando é representado pela Figura 2.24.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.55814/n1Host.conf# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   2300    560 ?        S    19:04   0:00 /usr/sbin/vnode
root        22  0.0  0.2   3492   1784 pts/5    Ss   19:04   0:00 /bin/bash
root        54  0.0  0.3   6332   2832 ?        Ss   19:04   0:00 /usr/sbin/apach
www-data   56  0.0  0.2   6120   1940 ?        S    19:04   0:00 /usr/sbin/apach
www-data   58  0.0  0.3 229800  2292 ?        S1   19:04   0:00 /usr/sbin/apach
www-data   59  0.0  0.3 228776  2300 ?        S1   19:04   0:00 /usr/sbin/apach
root       115  0.0  0.1   2868   1060 pts/5    R+   19:05   0:00 ps aux
root@n1Host:/tmp/pycore.55814/n1Host.conf# █

```

Figura 2.24: listagem dos processos de n1Host, incluindo /usr/sbin/apache2.

5. Ainda no terminal de n1Host, verifique a escuta da porta 80 para o serviço HTTP Apache em todas as suas interfaces de rede. Utilize o comando:

```
# netstat -antup
```

O resultado do comando é representado pela Figura 2.25.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.55814/n1Host.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp6      0      0 :::80                  :::*                    LISTEN
54/apache2
root@n1Host:/tmp/pycore.55814/n1Host.conf#

```

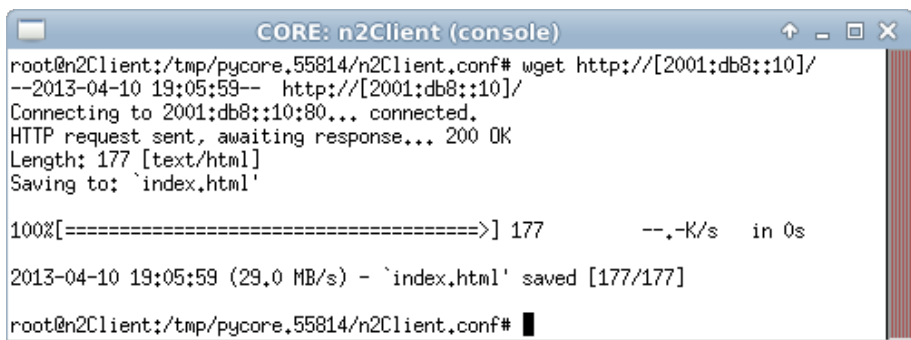
Figura 2.25: listagem das portas escutadas em n1Host, incluindo a TCP 80 em conexões IPv6.

Observe a *string* `:::80`. O endereço IPv6 `::/128` é utilizado para mostrar que o serviço não está atrelado a nenhum endereço IPv6 específico do dispositivo, isto é, o serviço pode ser acessado a partir de qualquer um dos endereços das interfaces de rede da máquina, enquanto que `:80` representa a porta 80, usualmente utilizada para receber requisições HTTP, por meio do protocolo TCP (IANA, 2014).

- Abra um terminal de n2Client com um duplo-clique e verifique o funcionamento do servidor HTTP n1Host ao realizar uma requisição HTTP GET:

```
# wget http://[2001:db8::10]/
```

O resultado do comando é representado pela Figura 2.26.



```

CORE: n2Client (console)
root@n2Client:/tmp/pycore.55814/n2Client.conf# wget http://[2001:db8::10]/
--2013-04-10 19:05:59-- http://[2001:db8::10]/
Connecting to 2001:db8::10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html'

100%[=====>] 177      --.-K/s  in 0s

2013-04-10 19:05:59 (29.0 MB/s) - `index.html' saved [177/177]

root@n2Client:/tmp/pycore.55814/n2Client.conf#

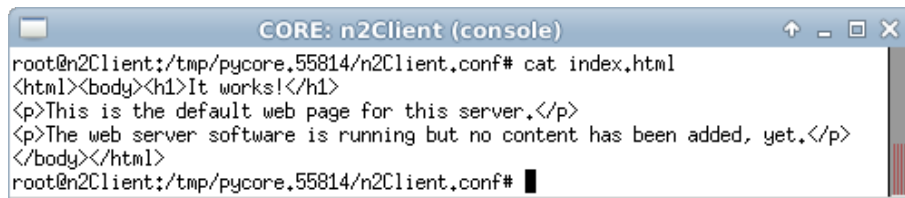
```

Figura 2.26: resultado esperado do acesso ao servidor HTTP oriundo de n2Client.

- Ainda no terminal de n2Client, verifique se o arquivo `index.html` foi transferido corretamente:

```
# cat index.html
```

O resultado do comando é representado pela Figura 2.27.



```

CORE: n2Client (console)
root@n2Client:/tmp/pycore.55814/n2Client.conf# cat index.html
<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
root@n2Client:/tmp/pycore.55814/n2Client.conf# █

```

Figura 2.27: resultado esperado do arquivo `index.html` transferido de `n1Host`.

- Abra um terminal de `n1Host` e verifique as entradas no `log` do Apache:

```
# cat /var/log/apache2/access.log
```

O resultado do comando é representado pela Figura 2.28.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.55814/n1Host.conf# cat /var/log/apache2/access.log
2001:db8::20 - - [10/Apr/2013:19:05:59 -0300] "GET / HTTP/1.1" 200 490 "-" "wget
/1.13.4 (linux-gnu)"
root@n1Host:/tmp/pycore.55814/n1Host.conf# █

```

Figura 2.28: verificação do `log` de acesso do Apache em `n1Host`.

Observe que o endereço registrado pelo servidor Apache é um endereço IPv6. Este fato é relevante, pois caso sejam utilizados *scripts* ou outras ferramentas de análise de `log`, tais ferramentas devem ser capazes de reconhecer corretamente o formato de endereços IPv6.

- Ainda no terminal de `n1Host`, pare o serviço HTTP Apache com o comando:

```
# /etc/init.d/apache2 stop
```

- Encerre a simulação, conforme descrito no Apêndice B.

Experiência 2.4. HTTP: habilitar IPv6 no Apache para páginas *Web* com configuração IPv4 existente

Objetivo

Na configuração padrão, o Apache aceita requisições enviadas a qualquer endereço IPv6 atribuído às interfaces do servidor. Neste laboratório, será trabalhada a utilização de um *VirtualHost*, que responderá a um endereço IPv6 específico, necessitando de configurações adicionais para isso. As configurações para o funcionamento com conexões IPv4 já estarão prontas.

Para a realização do presente exercício será utilizada a topologia descrita no arquivo: **2-04-HTTP-apache-preexistent.imn**.

Introdução teórica

Veja a introdução teórica em Experiência 2.3.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **2-04-HTTP-apache-preexistent.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 2.29, deve aparecer.

O objetivo desta topologia de rede é representar o mínimo necessário para que o serviço HTTP seja analisado.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós `n1Host` e `n2Client` e a conectividade entre eles.
3. Abra um terminal de `n2Client` com um duplo-clique e verifique o funcionamento do servidor HTTP `n1Host` realizando uma requisição HTTP GET por meio de uma conexão IPv4. Utilize o seguinte comando:

```
# wget 192.0.2.10
```

O resultado do comando é representado pela Figura 2.30.

Veja que o arquivo `index.html` foi transferido corretamente.

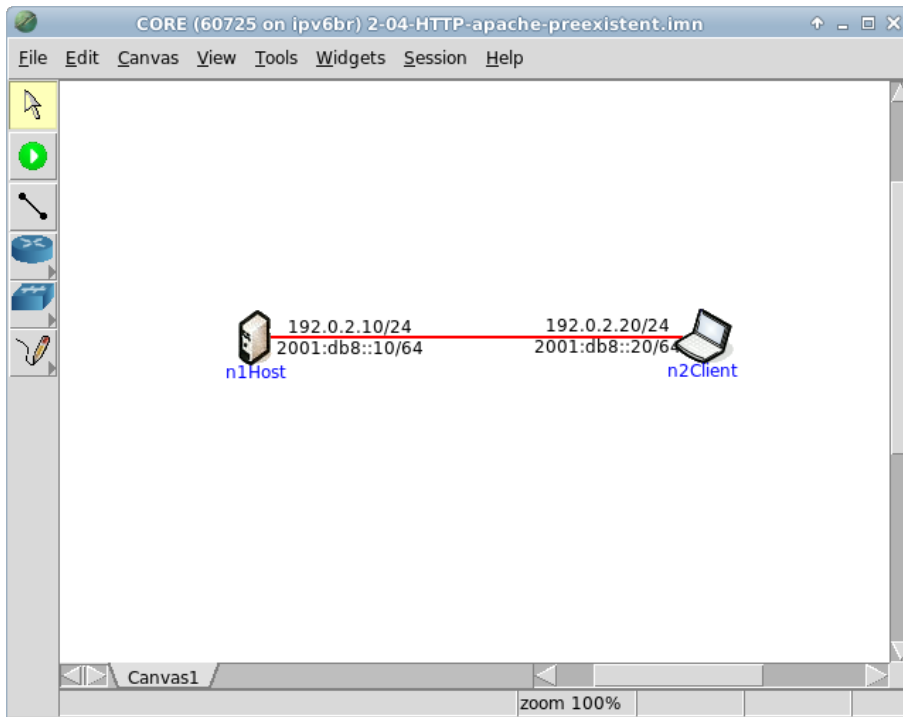


Figura 2.29: *topologia da Experiência 2.4 no CORE.*

```

CORE: n2Client (console)
root@n2Client:/tmp/pycore.55816/n2Client.conf# wget 192.0.2.10
--2013-04-10 19:08:06-- http://192.0.2.10/
Connecting to 192.0.2.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html'

100%[=====>] 177      --.-K/s   in 0s

2013-04-10 19:08:06 (31.4 MB/s) - `index.html' saved [177/177]
root@n2Client:/tmp/pycore.55816/n2Client.conf# █

```

Figura 2.30: *resultado do acesso ao servidor HTTP.*

4. Ainda no terminal de `n2Client`, verifique o funcionamento do servidor HTTP `n1Host` realizando uma requisição HTTP GET por meio de uma conexão IPv6:

```
# wget http://[2001:db8::10]/
```

O resultado do comando é representado pela Figura 2.31.



```
root@n2Client:/tmp/pycore.55816/n2Client.conf# wget http://[2001:db8::10]/
--2013-04-10 19:08:56-- http://[2001:db8::10]/
Connecting to 2001:db8::10:80... failed: Connection refused.
root@n2Client:/tmp/pycore.55816/n2Client.conf# █
```

Figura 2.31: resultado do acesso ao servidor HTTP.

Observe que o acesso por meio de uma conexão IPv6 foi recusado.

- Abra um terminal de n1Host com um duplo-clique e verifique os serviços ativos por meio do comando:

```
# netstat -antup
```

O resultado do comando é representado pela Figura 2.32.



```
root@n1Host:/tmp/pycore.55816/n1Host.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 192.0.2.10:80          0.0.0.0:*               LISTEN
43/apache2
root@n1Host:/tmp/pycore.55816/n1Host.conf# █
```

Figura 2.32: listagem das portas escutadas em n1Host.

Note que a porta 80 de n1Host, que recebe requisições HTTP (IANA, 2014), só recebe requisições para o endereço IPv4.

- Ainda no terminal de n1Host, edite o arquivo de configuração de página Web, localizado em `/etc/apache2/ports.conf`. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano.

Localize as seguintes linhas no arquivo:

```
NameVirtualHost 192.0.2.10:80
Listen 192.0.2.10:80
```

Logo abaixo da configuração do endereço IPv4, adicione as linhas:

```
NameVirtualHost [2001:db8::10]:80
Listen [2001:db8::10]:80
```

O endereço IPv6 deve estar entre colchetes, para diferenciar os campos do endereço da porta utilizada pelo protocolo HTTP para receber requisições.

Atente para o fato de que a especificação de endereços IP é bastante comum em servidores que se encontram em produção, com mais de uma página *Web* configurada. Nesses casos, para que o mesmo seja visto também em IPv6, os novos endereços devem ser adicionados na configuração, conforme explicitado neste passo.

7. Ainda no terminal de `n1Host`, edite o arquivo de configuração relativo ao *VirtualHost* configurado no passo 6, localizado em `/etc/apache2/sites-available/default`.

Adicione o trecho em **negrito** na linha referente ao *VirtualHost*, conforme apresentado a seguir:

```
<VirtualHost 192.0.2.10:80 [2001:db8::10]:80>
```

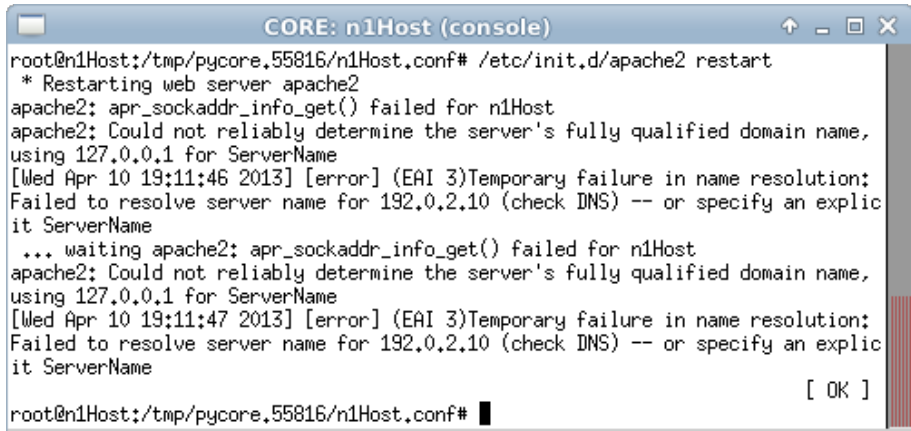
No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

8. Ainda no terminal de `n1Host`, reinicie o serviço HTTP Apache:

```
# /etc/init.d/apache2 restart
```

O resultado do comando é representado pela Figura 2.33.

Os avisos mostrados ocorrem pelo fato dos nomes de domínio e do servidor não estarem configurados adequadamente para a máquina. Entretanto, tais mensagens não afetam o funcionamento do serviço para a experiência.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.55816/n1Host.conf# /etc/init.d/apache2 restart
* Restarting web server apache2
apache2: apr_sockaddr_info_get() failed for n1Host
apache2: Could not reliably determine the server's fully qualified domain name,
using 127.0.0.1 for ServerName
[Wed Apr 10 19:11:46 2013] [error] (EAI 3)Temporary failure in name resolution:
Failed to resolve server name for 192.0.2.10 (check DNS) -- or specify an explic
it ServerName
... waiting apache2: apr_sockaddr_info_get() failed for n1Host
apache2: Could not reliably determine the server's fully qualified domain name,
using 127.0.0.1 for ServerName
[Wed Apr 10 19:11:47 2013] [error] (EAI 3)Temporary failure in name resolution:
Failed to resolve server name for 192.0.2.10 (check DNS) -- or specify an explic
it ServerName
[ OK ]
root@n1Host:/tmp/pycore.55816/n1Host.conf# █

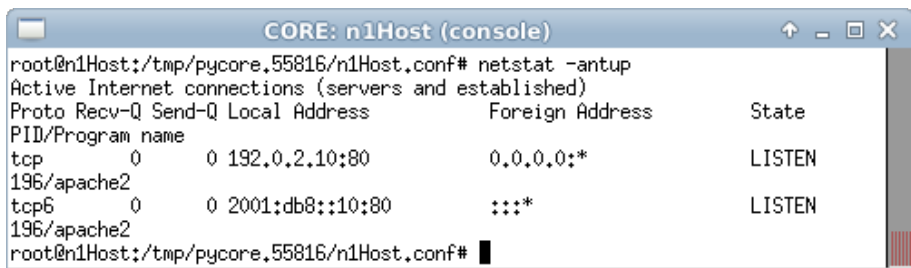
```

Figura 2.33: resultado da reinicialização do serviço HTTP Apache em n1Host.

- Ainda no terminal de n1Host, verifique a escuta da porta 80 em IPv4 e em IPv6. Utilize o seguinte comando:

```
# netstat -antup
```

O resultado do comando é representado pela Figura 2.34.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.55816/n1Host.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 192.0.2.10:80          0.0.0.0:*               LISTEN
196/apache2
tcp6       0      0 2001:db8::10:80       :::*                   LISTEN
196/apache2
root@n1Host:/tmp/pycore.55816/n1Host.conf# █

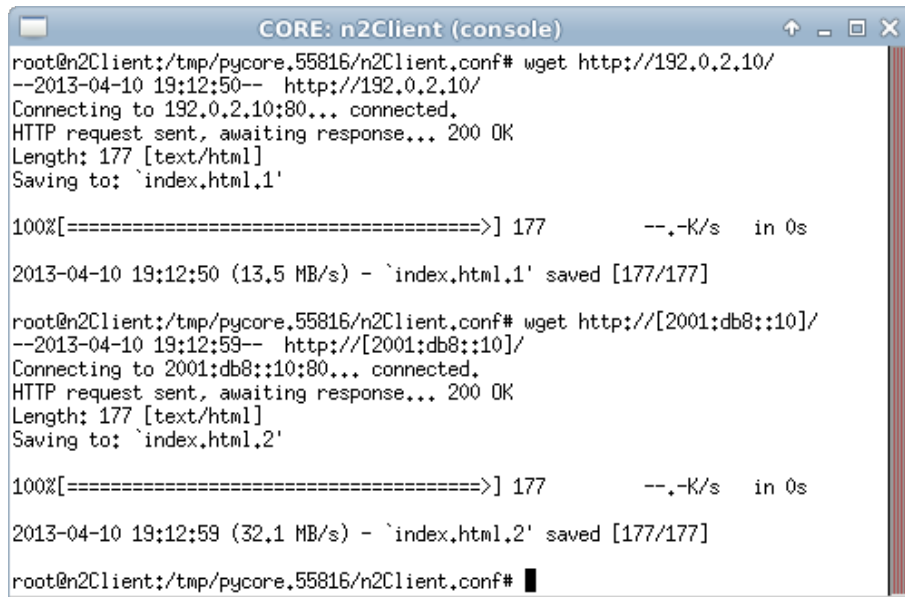
```

Figura 2.34: listagem das portas escutadas em n1Host, incluindo a TCP 80 em conexões IPv6.

- Abra um terminal de n2Client com um duplo-clique e verifique o funcionamento do servidor HTTP ao realizar requisições HTTP GET em IPv4 e em IPv6. Para isto digite os seguintes comandos:

```
# wget http://192.0.2.10/
# wget http://[2001:db8::10]/
```

O resultado do comando é representado pela Figura 2.35.



```
CORE: n2Client (console)
root@n2Client:/tmp/pycore.55816/n2Client.conf# wget http://192.0.2.10/
--2013-04-10 19:12:50-- http://192.0.2.10/
Connecting to 192.0.2.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html.1'

100%[=====>] 177      --.-K/s  in 0s

2013-04-10 19:12:50 (13,5 MB/s) - `index.html.1' saved [177/177]

root@n2Client:/tmp/pycore.55816/n2Client.conf# wget http://[2001:db8::10]/
--2013-04-10 19:12:59-- http://[2001:db8::10]/
Connecting to 2001:db8::10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html.2'

100%[=====>] 177      --.-K/s  in 0s

2013-04-10 19:12:59 (32,1 MB/s) - `index.html.2' saved [177/177]

root@n2Client:/tmp/pycore.55816/n2Client.conf#
```

Figura 2.35: resultado do acesso ao servidor HTTP por meio de conexões IPv4 e IPv6.

11. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 2.5. HTTP: configuração IPv6 no Nginx para novas páginas *Web*

Objetivo

O servidor *Web* Nginx, já apresenta suporte ao IPv6, porém este não vem habilitado por padrão. Neste laboratório, será apresentada a configuração básica para que o Nginx passe a aceitar requisições HTTP enviadas a qualquer endereço IPv6 atribuído às interfaces do servidor.

Para a realização do presente exercício será utilizada a topologia descrita no arquivo: **2-05-HTTP-nginx.imn**.

Introdução teórica

Veja a introdução teórica da Experiência 2.3.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **2-05-HTTP-nginx.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 2.36, deve aparecer.

O objetivo dessa topologia de rede é representar o mínimo necessário para que o serviço HTTP seja analisado.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós `n1Host` e `n2Client` e a conectividade entre eles.
3. Abra um terminal de `n1Host` e verifique o funcionamento do serviço Nginx:

```
# ps aux
```

O resultado do comando é representado pela Figura 2.37.

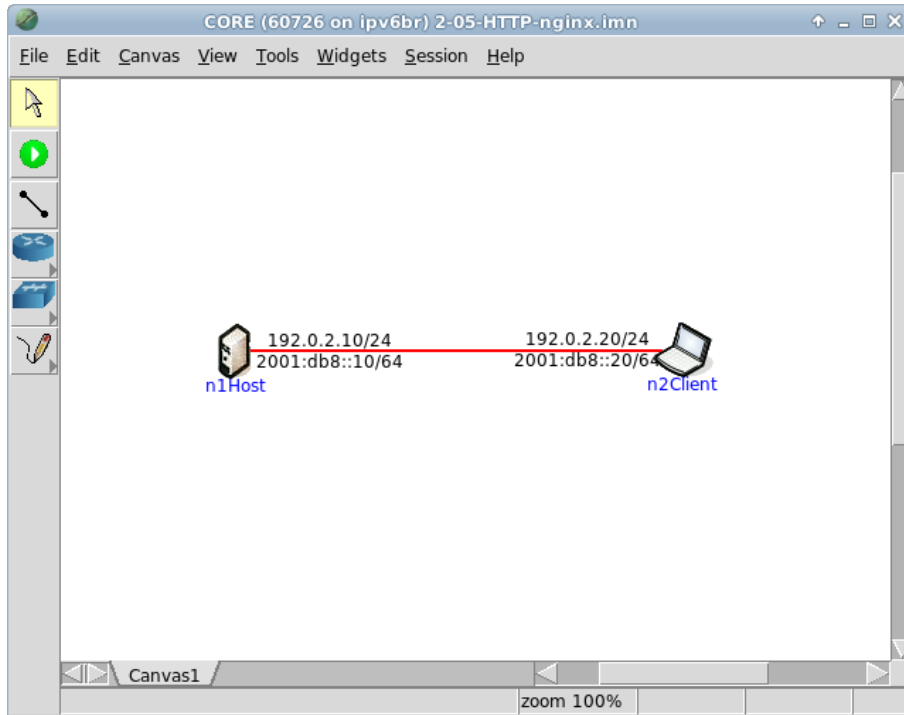


Figura 2.36: topologia da Experiência 2.5 no CORE.

```

root@n1Host:/tmp/pycore.33486/n1Host.conf# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   2300   556 ?        S      19:58   0:00 /usr/sbin/vnode
root        30  0.0  0.1  11976   948 ?        Ss     19:58   0:00 nginx: master p
www-data   31  0.0  0.1  12116  1364 ?        S      19:58   0:00 nginx: worker p
www-data   32  0.0  0.1  12116  1364 ?        S      19:58   0:00 nginx: worker p
www-data   33  0.0  0.1  12116  1364 ?        S      19:58   0:00 nginx: worker p
www-data   34  0.0  0.1  12116  1364 ?        S      19:58   0:00 nginx: worker p
root        35  0.5  0.2   3492   1744 pts/5    Ss     19:58   0:00 /bin/bash
root        46  0.0  0.1   2868   1056 pts/5    R+    19:59   0:00 ps aux
root@n1Host:/tmp/pycore.33486/n1Host.conf# █

```

Figura 2.37: listagem dos processos de n1Host, incluindo nginx.

4. Ainda no terminal de n1Host, verifique a escuta da porta 80 para o serviço HTTP Nginx em todas as suas interfaces de rede:

```
# netstat -antup
```

O resultado do comando é representado pela Figura 2.38.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.33486/n1Host.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
30/nginx
root@n1Host:/tmp/pycore.33486/n1Host.conf# █

```

Figura 2.38: listagem das portas escutadas em n1Host.

Note que ao contrário do observado sobre o Apache na Experiência 2.3, por padrão o Nginx não escuta a porta 80 em endereços IPv6, escutando somente a porta 80 em endereços IPv4.

5. Ainda no terminal de n1Host, edite o arquivo de configuração do Nginx, localizado em `/etc/nginx/sites-available/default`, de modo a descomentar as linhas 21 e 22, mostradas a seguir, removendo o caractere `#` do início da linha:

```

#listen 80; ## listen for ipv4; default and implied
#listen [::]:80 default ipv6only=on; ## listen for ipv6

```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano.

6. Ainda no terminal de n1Host, reinicie o serviço HTTP Nginx por meio do comando:

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

O resultado do comando é representado pela Figura 2.39.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.33486/n1Host.conf# /etc/init.d/nginx restart
Restarting nginx: nginx.
root@n1Host:/tmp/pycore.33486/n1Host.conf# █

```

Figura 2.39: resultado da reinicialização do serviço HTTP Nginx em n1Host.

- Ainda no terminal de `n1Host`, verifique a escuta da porta 80 em IPv4 e em IPv6. Para isto, execute o seguinte comando:

```
# netstat -antup
```

O resultado do comando é representado pela Figura 2.40.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.33486/n1Host.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:80             0.0.0.0:*              LISTEN
72/nginx
tcp6       0      0 :::80                  :::*                    LISTEN
72/nginx
root@n1Host:/tmp/pycore.33486/n1Host.conf#

```

Figura 2.40: listagem das portas escutadas em `n1Host`, incluindo a TCP 80 em conexões IPv6.

Observe a *string* `:::80`. O endereço IPv6 `::/128` é utilizado para mostrar que o serviço não está atrelado a nenhum endereço IPv6 específico do dispositivo, isto é, o serviço pode ser acessado a partir de qualquer um dos endereços das interfaces de rede da máquina, enquanto que `:80` representa a porta 80, usualmente utilizada para receber requisições HTTP, por meio do protocolo TCP (IANA, 2014).

- Abra um terminal de `n2Client` com um duplo-clique e verifique o funcionamento do servidor HTTP ao realizar requisições HTTP GET em IPv4 e em IPv6:

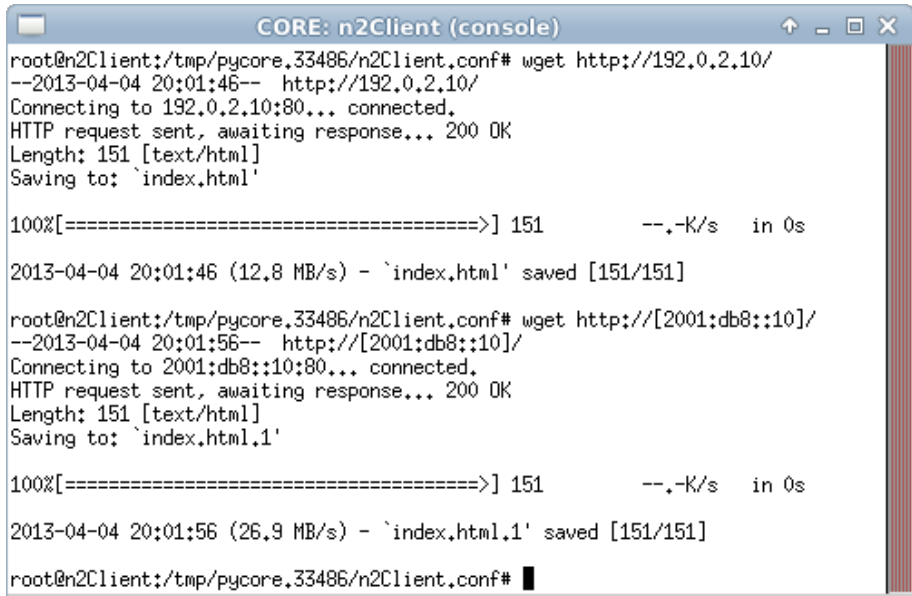
```
# wget http://192.0.2.10/
# wget http://[2001:db8::10]/
```

O resultado do comando é representado pela Figura 2.41.

9. Ainda no terminal de n2Client, verifique se os arquivos `index.html` e `index.html.1` foram transferidos corretamente por meio dos comandos:

```
# diff index.html index.html.1
# cat index.html
```

O resultado dos comandos é representado pela Figura 2.42.



```
CORE: n2Client (console)
root@n2Client:/tmp/pycore.33486/n2Client.conf# wget http://192.0.2.10/
--2013-04-04 20:01:46-- http://192.0.2.10/
Connecting to 192.0.2.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 151 [text/html]
Saving to: `index.html'

100%[=====>] 151      --.-K/s  in 0s

2013-04-04 20:01:46 (12,8 MB/s) - `index.html' saved [151/151]

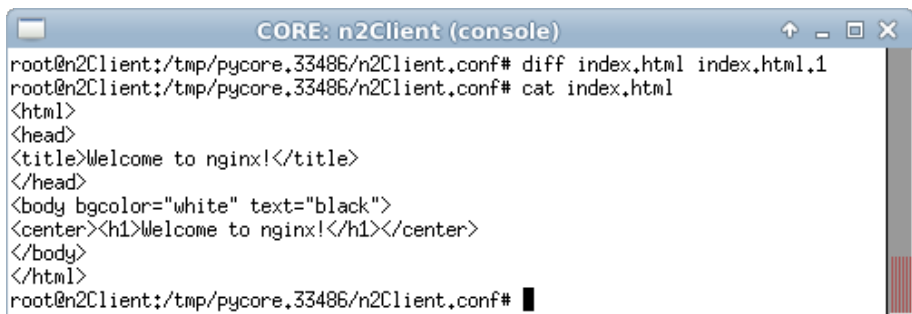
root@n2Client:/tmp/pycore.33486/n2Client.conf# wget http://[2001:db8::10]/
--2013-04-04 20:01:56-- http://[2001:db8::10]/
Connecting to 2001:db8::10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 151 [text/html]
Saving to: `index.html.1'

100%[=====>] 151      --.-K/s  in 0s

2013-04-04 20:01:56 (26,9 MB/s) - `index.html.1' saved [151/151]

root@n2Client:/tmp/pycore.33486/n2Client.conf# █
```

Figura 2.41: resultado do acesso ao servidor HTTP por meio de conexões IPv4 e do IPv6.



```
CORE: n2Client (console)
root@n2Client:/tmp/pycore.33486/n2Client.conf# diff index.html index.html.1
root@n2Client:/tmp/pycore.33486/n2Client.conf# cat index.html
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Welcome to nginx!</h1></center>
</body>
</html>
root@n2Client:/tmp/pycore.33486/n2Client.conf# █
```

Figura 2.42: resultado esperado dos arquivos transferidos de n1Host.

Veja que os arquivos são iguais uma vez que apenas um *VirtualServer* foi configurado no Nginx.

10. Abra um terminal de n1Host com um duplo-clique e execute o seguinte comando para verificar as entradas no *log* do Nginx:

```
# cat /var/log/nginx/access.log
```

O resultado do comando é representado pela Figura 2.43.



```
root@n1Host:/tmp/pycore.33486/n1Host.conf# cat /var/log/nginx/access.log
192.0.2.20 - - [04/Apr/2013:20:01:46 -0300] "GET / HTTP/1.1" 200 151 "-" "wget/1
.13.4 (linux-gnu)"
2001:db8::20 - - [04/Apr/2013:20:01:56 -0300] "GET / HTTP/1.1" 200 151 "-" "wget
/1.13.4 (linux-gnu)"
root@n1Host:/tmp/pycore.33486/n1Host.conf# █
```

Figura 2.43: verificação do log de acesso do Nginx em n1Host.

O servidor Nginx registra tanto endereços IPv4 quanto endereços IPv6. Este fato é relevante, pois caso sejam utilizados *scripts* ou outras ferramentas de análise de *log*, tais ferramentas devem ser capazes de reconhecer corretamente o formato de endereços IPv6.

11. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 2.6. *Proxy*: configuração de *Proxy Web* direto em rede IPv6

Objetivo

Esta experiência tem o objetivo de mostrar a configuração de um *proxy Web* utilizando o *software* Squid. A primeira configuração permitirá o acesso à Internet IPv4 por dispositivos em uma rede que opera somente com IPv6. Outro exemplo será apresentado mostrando o cenário oposto, em que um dispositivo em uma rede IPv4 acessará o conteúdo de uma rede IPv6.

Para a realização do presente exercício será utilizada a topologia descrita no arquivo: **2-06-proxy-forward.imn**.

Introdução teórica

Um *proxy* pode ser utilizado em redes corporativas para intermediar o acesso à *Web*. Algumas de suas funções são:

- Melhorar a velocidade de acesso à *Web* e diminuir a quantidade de dados trafegados no enlace Internet, com a utilização de *cache*.
- Controlar acesso a conteúdos ou serviços específicos.
- Manter registro dos acessos a páginas *Web*.
- Verificar a existência de *malware* antes de entregar um conteúdo requisitado.
- Manter dispositivos anônimos, por questões de segurança.

Durante a implantação do IPv6 em uma rede, servidores *proxy* podem também ser utilizados como mecanismos de transição. Um *proxy* configurado com pilha dupla pode permitir o acesso à *Web* por meio do IPv6 oriundo de um dispositivo IPv4, e vice-versa.

O Squid, um dos servidores *proxy* mais utilizados para plataforma Linux, suporta IPv6 desde sua versão 2.6 por meio de um *patch* específico e nativamente desde a versão 3.1. Servidores *Web* como o Apache, o Nginx ou o Microsoft IIS Server também podem ser utilizados nessa função.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **2-06-proxy-forward.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 2.44, deve aparecer.

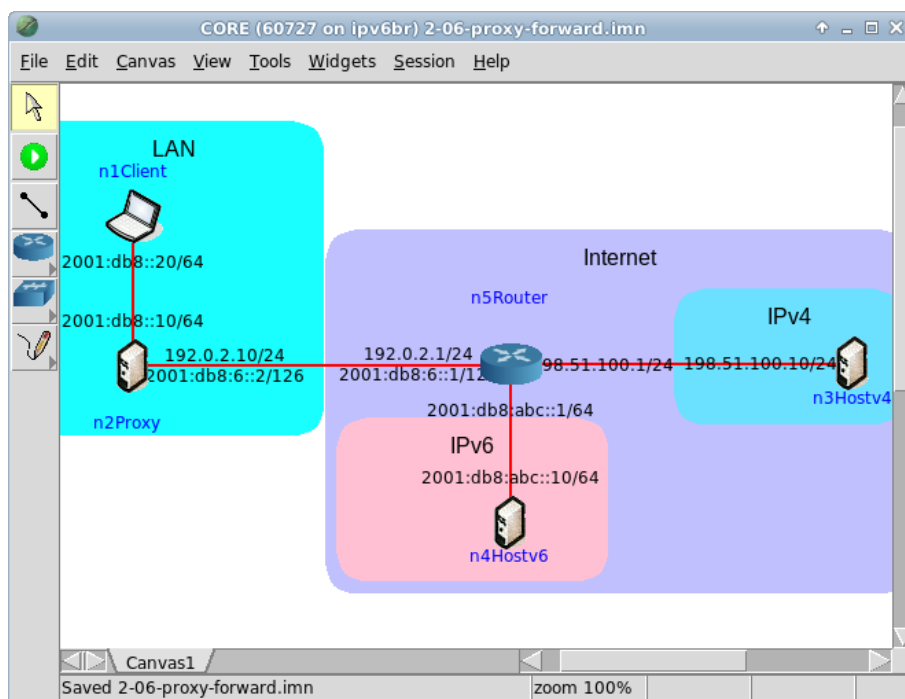


Figura 2.44: topologia da *Experiência 2.6* no CORE.

Esta topologia representa uma situação em que a rede local de um cliente possui apenas IPv6 e, para acessar conteúdo *Web* disponível em endereços IPv4, um *proxy* é utilizado. Nela, o roteamento de pacotes foi configurado por meio de rotas estáticas. O nó *n2Proxy*, além de suas funções básicas, funciona também como um roteador da rede local, com o intuito de simplificar a topologia.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 em todos os nós e a conectividade entre eles.
 3. Verifique o funcionamento do serviço HTTP Apache nos servidores.
- (a) Abra um terminal de n3Hostv4 com um duplo-clique e verifique a escuta da porta 80. Utilize o comando:

```
# netstat -antup
```

O resultado do comando é representado pela Figura 2.45.



```

CORE: n3Hostv4 (console)
root@n3Hostv4:/tmp/pycore.55826/n3Hostv4.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
35/apache2
root@n3Hostv4:/tmp/pycore.55826/n3Hostv4.conf#

```

Figura 2.45: listagem das portas escutadas em n3Hostv4, incluindo a TCP 80 em conexões IPv4.

- (b) Abra um terminal de n4Hostv6 com um duplo-clique e verifique a escuta da porta 80. Execute o seguinte comando:

```
# netstat -antup
```

O resultado do comando é representado pela Figura 2.46.



```

CORE: n4Hostv6 (console)
root@n4Hostv6:/tmp/pycore.55826/n4Hostv6.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp6       0      0 :::80                   :::*                    LISTEN
35/apache2
root@n4Hostv6:/tmp/pycore.55826/n4Hostv6.conf#

```

Figura 2.46: listagem das portas escutadas em n4Hostv6, incluindo a TCP 80 em conexões IPv6.

Observe a *string* `:::80`. O endereço IPv6 `::/128` é utilizado para mostrar que o serviço não está atrelado a nenhum endereço IPv6 específico do dispositivo, isto é, o serviço pode ser acessado a partir de qualquer um dos endereços das interfaces de rede da máquina, enquanto que `:80` representa a porta 80, usualmente utilizada para receber requisições HTTP, por meio do protocolo TCP (IANA, 2014).

4. Abra um terminal de `n2Proxy` com um duplo-clique e verifique o conteúdo do arquivo de configuração do Squid. Para isto utilize comando:

```
# cat /etc/squid3/squid.conf
```

O arquivo deverá conter as linhas:

```
acl manager proto cache_object
acl localhost src 127.0.0.1/32 ::1
acl to_localhost dst 127.0.0.0/8 0.0.0.0/32 ::1
acl localnet src 2001:db8::/64
acl SSL_ports port 443
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443         # https
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535 # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 777         # multiling http
acl CONNECT method CONNECT
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow localnet
http_access allow localhost
```

```

http_access deny all
icp_access deny all
htcp_access deny all
http_port 3128
hierarchy_stoplister cgi-bin ?
access_log /var/log/squid3/access.log squid
refresh_pattern ^ftp: 1440 20% 10080
refresh_pattern ^gopher: 1440 0% 1440
refresh_pattern (cgi-bin|\\?) 0 0% 0
refresh_pattern . 0 20% 4320
icp_port 3130
coredump_dir /var/spool/squid3

```

O serviço permitirá, por meio do comando `http_access allow localnet`, acesso da rede local que foi definida pelo prefixo `2001:db8::/64` na linha `acl localnet src 2001:db8::/64`. A linha de configuração `http_port 3128` é utilizada para definir que o serviço Squid deve escutar a porta 3128, usualmente utilizada para receber requisições por meio do *proxy* (IANA, 2014).

5. Ainda no terminal de `n2Proxy`, utilize o seguinte comando para iniciar o serviço Squid:

```
# squid3 -s -f /etc/squid3/squid.conf
```

O resultado do comando é representado pela Figura 2.47.



Figura 2.47: resultado da inicialização do serviço de proxy Squid.

6. Ainda no terminal de `n2Proxy`, verifique a escuta da porta 3128. Para isto digite o seguinte comando:

```
# netstat -antup
```

```

CORE: n2Proxy (console)
root@n2Proxy:/tmp/pycore.55826/n2Proxy.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp6      0      0 :::3128                :::*                    LISTEN
60/(squid)
udp       0      0 0.0.0.0:59508          0.0.0.0:*
60/(squid)
udp6      0      0 :::3130                :::*
60/(squid)
udp6      0      0 :::40556               :::*
60/(squid)
root@n2Proxy:/tmp/pycore.55826/n2Proxy.conf#

```

Figura 2.48: listagem das portas escutadas em n2Proxy, incluindo a TCP 3128 em conexões IPv6.

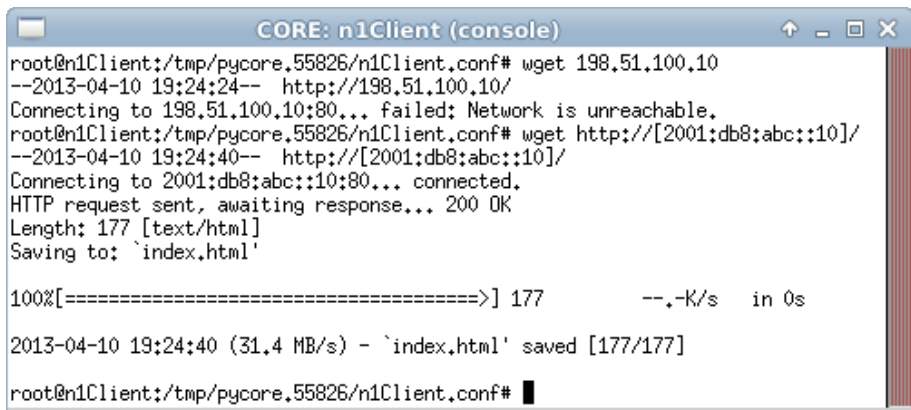
O resultado do comando deve ser similar à Figura 2.48.

A porta UDP 3130 foi configurada para o serviço ICP (*Internet Cache Protocol*), útil em casos de comunicação entre servidores *proxy*. As portas UDP listadas são escolhidas aleatoriamente por padrão e são utilizadas com os módulos ICP, HTCP e DNS do Squid.

7. Configure o `n1Client` para utilizar `n2Proxy` para acessar páginas *Web* disponíveis apenas em IPv4.
- (a) Abra um terminal de `n1Client` com um duplo-clique e verifique a acessibilidade do serviço HTTP de `n3Hostv4` e `n4Hostv6` por meio dos comandos:

```
# wget 198.51.100.10
# wget http://[2001:db8:abc::10]/
```

O resultado dos comandos é representado pela Figura 2.49.



```
CORE: n1Client (console)
root@n1Client:/tmp/pycore.55826/n1Client.conf# wget 198.51.100.10
--2013-04-10 19:24:24-- http://198.51.100.10/
Connecting to 198.51.100.10:80... failed: Network is unreachable.
root@n1Client:/tmp/pycore.55826/n1Client.conf# wget http://[2001:db8:abc::10]/
--2013-04-10 19:24:40-- http://[2001:db8:abc::10]/
Connecting to 2001:db8:abc::10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html'

100%[=====>] 177      --.-K/s   in 0s

2013-04-10 19:24:40 (31.4 MB/s) - `index.html' saved [177/177]

root@n1Client:/tmp/pycore.55826/n1Client.conf# █
```

Figura 2.49: acesso do serviço HTTP somente em IPv6 oriundo de `n1Client`.

Note que somente `n4Hostv6` está acessível a `n1Client`.

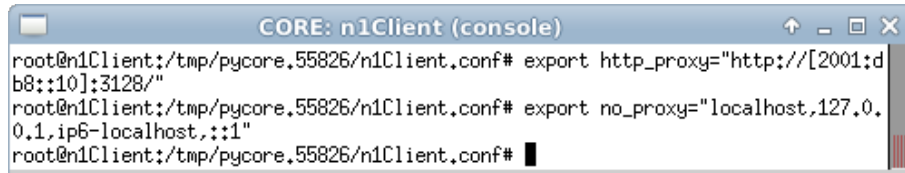
- (b) Ainda no terminal de `n1Client`, configure as variáveis de ambiente de forma que o *proxy* seja utilizado para acessar páginas *Web*. Para isto, os seguintes comandos precisam ser executados:

```
# export http_proxy="http://[2001:db8::10]:3128/"
# export no_proxy="localhost,127.0.0.1,ip6-localhost,::1"
```

O resultado dos comandos é representado pela Figura 2.50.

A configuração para utilização de *proxy* pode variar de aplicação para aplicação, sendo necessário consultar a documentação de cada uma para verificar a configuração correta. O exemplo anterior é válido apenas para algumas aplicações, entre elas o `wget`.

- (c) Ainda no terminal de `n1Client`, verifique novamente a acessibilidade do serviço HTTP de `n3Hostv4` e `n4Hostv6`. Utilize os comandos:



```

CORE: n1Client (console)
root@n1Client:/tmp/pycore.55826/n1Client.conf# export http_proxy="http://[2001:db8::10]:3128/"
root@n1Client:/tmp/pycore.55826/n1Client.conf# export no_proxy="localhost,127.0.0.1,ip6-localhost,::1"
root@n1Client:/tmp/pycore.55826/n1Client.conf#

```

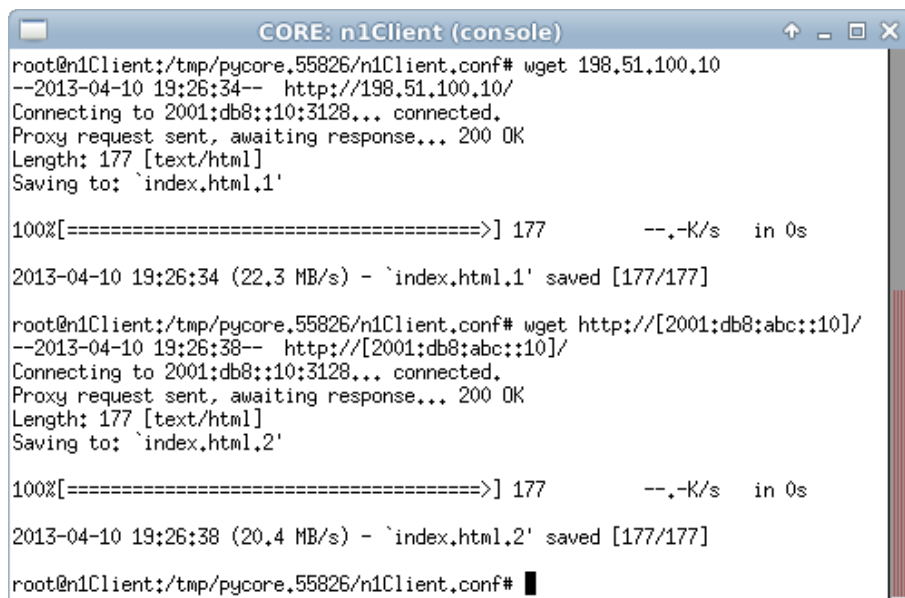
Figura 2.50: resultado da configuração do proxy IPv6 em n1Client.

```

# wget 198.51.100.10
# wget http://[2001:db8:abc::10]/

```

O resultado dos comandos é representado pela Figura 2.51.



```

CORE: n1Client (console)
root@n1Client:/tmp/pycore.55826/n1Client.conf# wget 198.51.100.10
--2013-04-10 19:26:34-- http://198.51.100.10/
Connecting to 2001:db8::10:3128... connected.
Proxy request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html.1'

100%[=====] 177      --.-K/s  in 0s

2013-04-10 19:26:34 (22.3 MB/s) - `index.html.1' saved [177/177]

root@n1Client:/tmp/pycore.55826/n1Client.conf# wget http://[2001:db8:abc::10]/
--2013-04-10 19:26:38-- http://[2001:db8:abc::10]/
Connecting to 2001:db8::10:3128... connected.
Proxy request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html.2'

100%[=====] 177      --.-K/s  in 0s

2013-04-10 19:26:38 (20.4 MB/s) - `index.html.2' saved [177/177]

root@n1Client:/tmp/pycore.55826/n1Client.conf#

```

Figura 2.51: acesso do serviço HTTP utilizando o proxy IPv6 oriundo de n1Client.

Em ambos os casos, o acesso é feito por meio do serviço de *proxy* que se encontra na porta 3128 do endereço IPv6 2001:db8::10.

8. Nos passos seguintes o cenário será o oposto ao que foi visto até o momento, de modo que `n1Client` será configurada para utilizar `n2Proxy` para acessar páginas *Web* disponíveis apenas em IPv6 utilizando IPv4.
- (a) Abra um terminal de `n2Proxy` com um duplo-clique e adicione um endereço IPv4 em sua interface de rede `eth0`. Digite os seguintes comandos:

```
# ip addr add 203.0.113.10/24 dev eth0
# ip addr show
```

O resultado dos comandos é representado pela Figura 2.52.

```

CORE: n2Proxy (console)
root@n2Proxy:/tmp/pycore.55826/n2Proxy.conf# ip addr add 203.0.113.10/24 dev eth0
root@n2Proxy:/tmp/pycore.55826/n2Proxy.conf# ip addr show
39: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
46: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP q
len 1000
    link/ether 00:00:00:aa:00:01 brd ff:ff:ff:ff:ff:ff
    inet 203.0.113.10/24 scope global eth0
        inet6 2001:db8::10/64 scope global
            valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feaa:1/64 scope link
        valid_lft forever preferred_lft forever
49: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP q
len 1000
    link/ether 00:00:00:aa:00:02 brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.10/24 scope global eth1
        inet6 2001:db8:6::2/126 scope global
            valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feaa:2/64 scope link
        valid_lft forever preferred_lft forever
root@n2Proxy:/tmp/pycore.55826/n2Proxy.conf# █

```

Figura 2.52: configuração de endereço IPv4 em `n2Proxy`.

- (b) Ainda no terminal de `n2Proxy`, edite o arquivo de configuração do Squid, localizado em `/etc/squid3/squid.conf` e adicione o trecho em **negrito** na linha referente às redes internas, conforme apresentado a seguir:

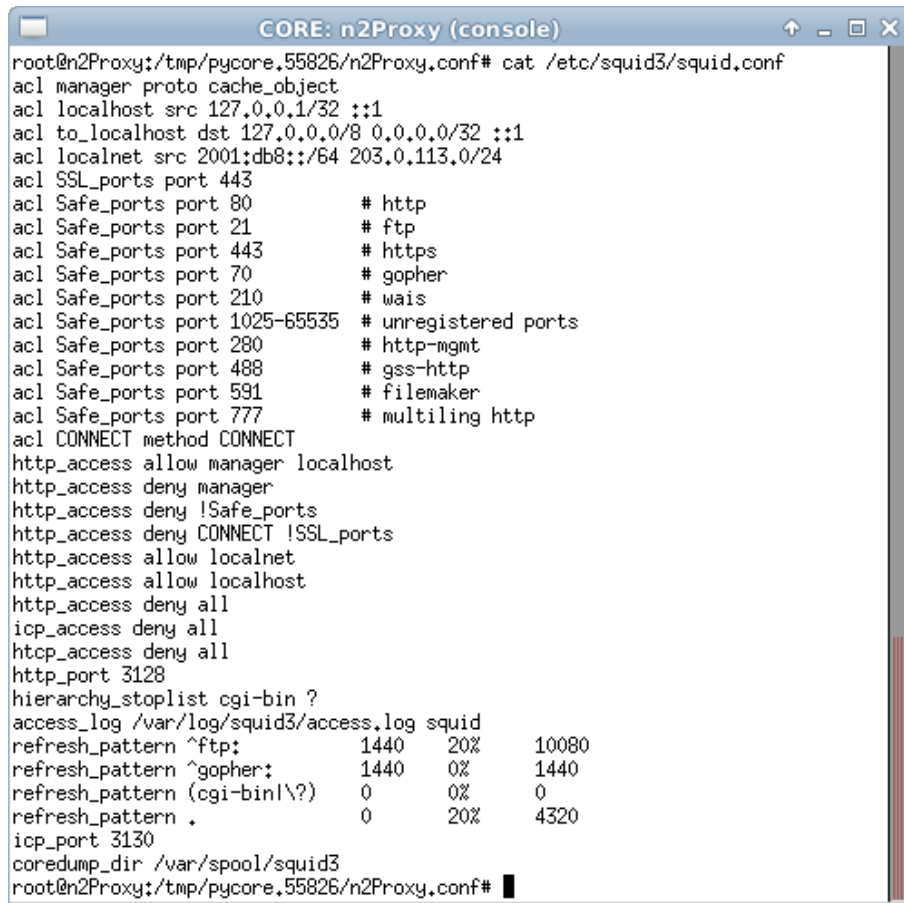
```
acl localnet src 2001:db8::/64 203.0.113.0/24
```

Desse modo, a rede IPv4 estabelecida será atendida pelo serviço de *proxy* Squid. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

Verifique o conteúdo do arquivo de configuração do Squid. Utilize o comando:

```
# cat /etc/squid3/squid.conf
```

O resultado da alteração é representado pela Figura 2.53.



```

CORE: n2Proxy (console)
root@n2Proxy:/tmp/pycore.55826/n2Proxy.conf# cat /etc/squid3/squid.conf
acl manager proto cache_object
acl localhost src 127.0.0.1/32 ::1
acl to_localhost dst 127.0.0.0/8 0.0.0.0/32 ::1
acl localnet src 2001:db8::/64 203.0.113.0/24
acl SSL_ports port 443
acl Safe_ports port 80          # http
acl Safe_ports port 21         # ftp
acl Safe_ports port 443       # https
acl Safe_ports port 70        # gopher
acl Safe_ports port 210       # wais
acl Safe_ports port 1025-65535 # unregistered ports
acl Safe_ports port 280       # http-mgmt
acl Safe_ports port 488       # gss-http
acl Safe_ports port 591       # filemaker
acl Safe_ports port 777       # multiling http
acl CONNECT method CONNECT
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow localnet
http_access allow localhost
http_access deny all
icp_access deny all
htcp_access deny all
http_port 3128
hierarchy_stoplist cgi-bin ?
access_log /var/log/squid3/access.log squid
refresh_pattern ^ftp:      1440 20% 10080
refresh_pattern ^gopher:  1440 0% 1440
refresh_pattern (cgi-bin|?) 0 0% 0
refresh_pattern .         0 20% 4320
icp_port 3130
coredump_dir /var/spool/squid3
root@n2Proxy:/tmp/pycore.55826/n2Proxy.conf#

```

Figura 2.53: verificação do arquivo de configuração do Squid após sua edição.

- (c) Ainda no terminal de n2Proxy, reinicialize o serviço de *proxy* Squid. Para isto digite:

```
# squid3 -k reconfigure
```

O resultado do comando é representado pela Figura 2.54.



```

CORE: n2Proxy (console)
root@n2Proxy:/tmp/pycore.55826/n2Proxy.conf# squid3 -k reconfigure
root@n2Proxy:/tmp/pycore.55826/n2Proxy.conf#

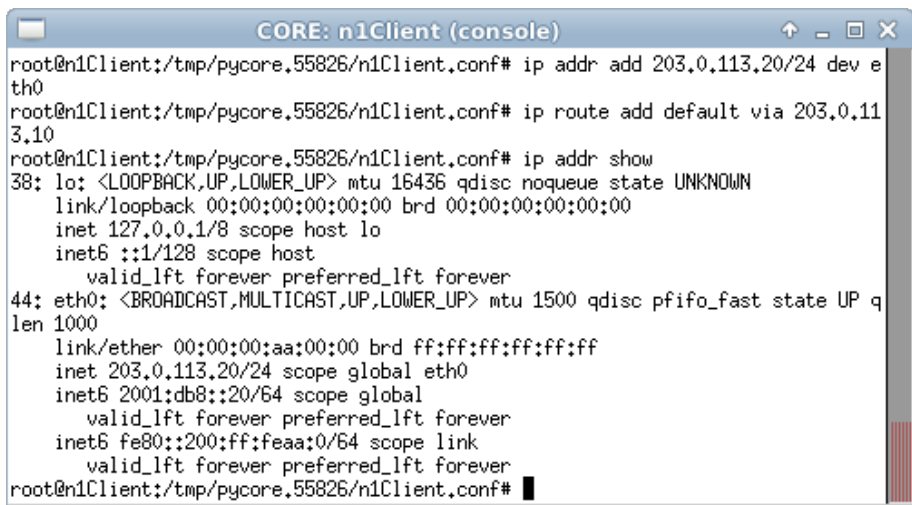
```

Figura 2.54: resultado da reconfiguração do serviço de *proxy* Squid.

- (d) Abra um terminal de n1Client com um duplo-clique e adicione um endereço IPv4 em sua interface de rede eth0. Utilize os seguintes comandos para isto:

```
# ip addr add 203.0.113.20/24 dev eth0
# ip route add default via 203.0.113.10
# ip addr show
```

O resultado dos comandos é representado pela Figura 2.55.



```

CORE: n1Client (console)
root@n1Client:/tmp/pycore.55826/n1Client.conf# ip addr add 203.0.113.20/24 dev e
th0
root@n1Client:/tmp/pycore.55826/n1Client.conf# ip route add default via 203.0.11
3.10
root@n1Client:/tmp/pycore.55826/n1Client.conf# ip addr show
38: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
44: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP q
len 1000
    link/ether 00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
    inet 203.0.113.20/24 scope global eth0
    inet6 2001:db8::20/64 scope global
        valid_lft forever preferred_lft forever
    inet6 fe80::200:ff:feaa:0/64 scope link
        valid_lft forever preferred_lft forever
root@n1Client:/tmp/pycore.55826/n1Client.conf#

```

Figura 2.55: configuração de endereço IPv4 em n1Client.

- (e) Ainda no terminal de `n1Client`, configure as variáveis de ambiente de forma que o *proxy* seja utilizado para acessar páginas *Web*. Para isto os seguintes comandos precisam ser executados:

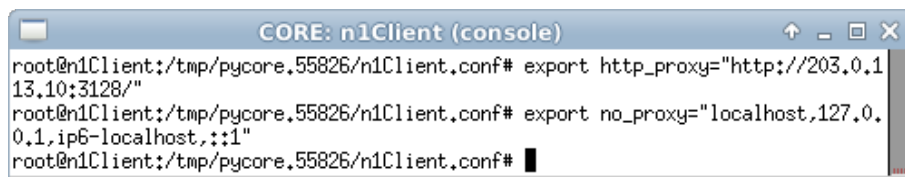
```
# export http_proxy="http://203.0.113.10:3128/"
# export no_proxy="localhost,127.0.0.1,ip6-localhost,::1"
```

O resultado do primeiro comando é representado pela Figura 2.56.

- (f) Ainda no terminal de `n1Client`, verifique a acessibilidade do serviço HTTP de `n3Hostv4` e `n4Hostv6` por meio do *proxy*. Digite os seguintes comandos:

```
# wget 198.51.100.10
# wget http://[2001:db8:abc::10]/
```

O resultado dos comandos é representado pela Figura 2.57.



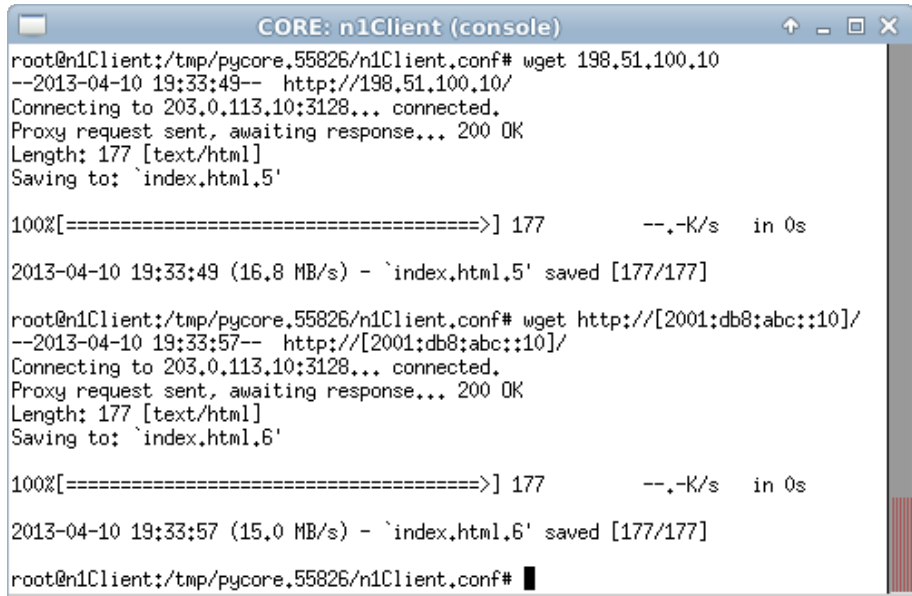
```
CORE: n1Client (console)
root@n1Client:/tmp/pycore.55826/n1Client.conf# export http_proxy="http://203.0.113.10:3128/"
root@n1Client:/tmp/pycore.55826/n1Client.conf# export no_proxy="localhost,127.0.0.1,ip6-localhost,::1"
root@n1Client:/tmp/pycore.55826/n1Client.conf# █
```

Figura 2.56: resultado da configuração do *proxy IPv4* em `n1Client`.

Note que, em ambos os casos, o acesso é feito por meio do serviço de *proxy*, que se encontra na porta 3128 do endereço IPv4 203.0.113.10.

9. Abra um terminal de `n2Proxy` com um duplo-clique e pare o serviço de *proxy* Squid. Execute o comando:

```
# squid3 -k shutdown
```



```
CORE: n1Client (console)
root@n1Client:/tmp/pycore.55826/n1Client.conf# wget 198.51.100.10
--2013-04-10 19:33:49-- http://198.51.100.10/
Connecting to 203.0.113.10:3128... connected.
Proxy request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html.5'

100%[=====>] 177      --.-K/s  in 0s

2013-04-10 19:33:49 (16,8 MB/s) - `index.html.5' saved [177/177]

root@n1Client:/tmp/pycore.55826/n1Client.conf# wget http://[2001:db8:abc::10]/
--2013-04-10 19:33:57-- http://[2001:db8:abc::10]/
Connecting to 203.0.113.10:3128... connected.
Proxy request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html.6'

100%[=====>] 177      --.-K/s  in 0s

2013-04-10 19:33:57 (15,0 MB/s) - `index.html.6' saved [177/177]

root@n1Client:/tmp/pycore.55826/n1Client.conf# █
```

Figura 2.57: acesso do serviço HTTP utilizando o proxy IPv4 oriundo de n1Client.

10. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 2.7. *Proxy Web* reverso: configuração de *proxy Web em rede de servidores*

Objetivo

Esta experiência tem o objetivo de mostrar a configuração básica de um *proxy* reverso configurado para permitir o acesso de clientes oriundos da Internet a uma página *Web* alocado em uma rede que opera somente IPv6. Será possível observar um cliente efetuando requisições usando IPv4 e o servidor respondendo-as em IPv6, com o *proxy* funcionando como *cache* e tradutor.

Para a realização do presente exercício será utilizada a topologia descrita no arquivo: **2-07-proxy-reverse.imn**.

Introdução teórica

Um *proxy* reverso pode funcionar como *frontend* para servidores *Web*, com diversas finalidades. Por exemplo, *cache* das informações, segurança do servidor *Web*, balanceamento de carga, etc. Os clientes interagem apenas com o *proxy*, não tendo acesso direto ao servidor *Web* original.

Os *proxies* reversos podem ser usados como um mecanismo de transição. Se configurados com pilha dupla, podem permitir que um cliente IPv6 acesse um servidor *Web* que usa apenas IPv4 ou vice versa.

Vários balanceadores de carga comerciais oferecem essa funcionalidade. É possível implementá-la também usando Squid, Apache, Nginx e outros *software*.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **2-07-proxy-reverse.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 2.58, deve aparecer.

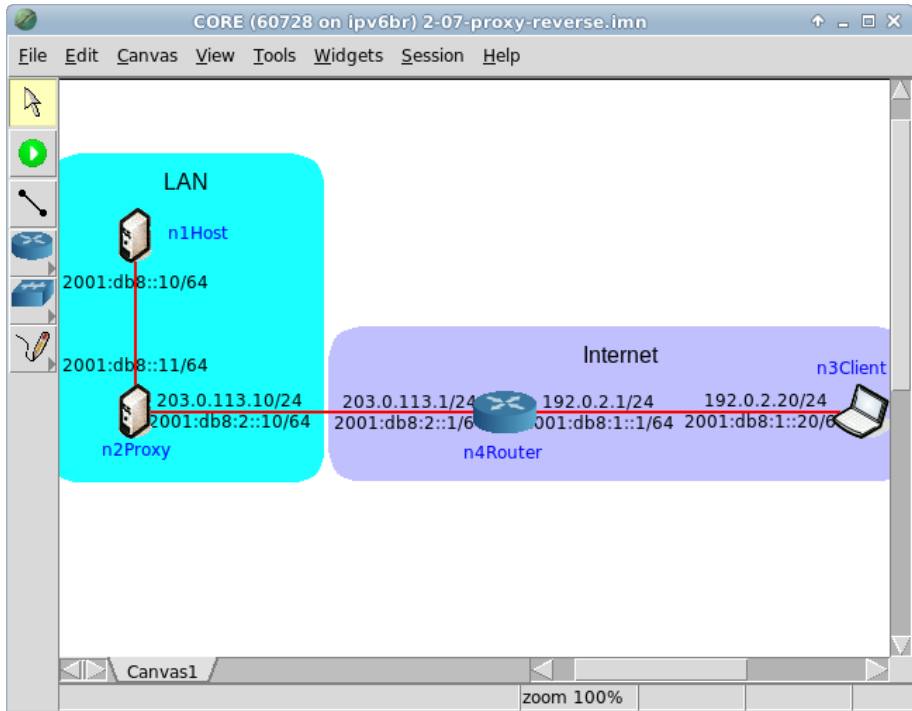


Figura 2.58: topologia da *Experiência 2.7* no CORE.

Esta topologia representa uma situação em que um *proxy* reverso é configurado como intermediário entre uma rede IPv6 e a Internet, funcionando como *cache* transparente para um servidor *Web* e traduzindo requisições IPv4 realizadas. O roteamento de pacotes foi configurado por meio de rotas estáticas. O nó *n2Proxy* funciona também como um roteador para manter a topologia simples.

2. Conforme descrito nos Apêndices B e C, inicie a simulação, verifique a configuração de endereços IPv6 em todos os nós e a conectividade entre eles.
3. Abra um terminal de *n1Host* com um duplo-clique e verifique a escuta da porta 80:

```
# netstat -antup
```

O resultado do comando é representado pela Figura 2.59.



```

CORE: n1Host (console)
root@n1Host:/tmp/pycore.55840/n1Host.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp6      0      0 :::80                   :::*                    LISTEN
35/apache2
root@n1Host:/tmp/pycore.55840/n1Host.conf# █

```

Figura 2.59: listagem das portas escutadas em n1Host, incluindo a TCP 80 em conexões IPv6.

Observe a *string* `:::80`. O endereço IPv6 `::/128` é utilizado para mostrar que o serviço não está atrelado a nenhum endereço IPv6 específico do dispositivo. Isto é, o serviço pode ser acessado a partir de qualquer um dos endereços das interfaces de rede da máquina, enquanto que `:80` representa a porta 80, usualmente utilizada para receber requisições HTTP, por meio do protocolo TCP (IANA, 2014).

4. Abra um terminal de n2Proxy com um duplo-clique e verifique o conteúdo do arquivo de configuração do Squid:

```
# cat /etc/squid3/squid.conf
```

O arquivo deverá conter as linhas:

```

http_port 80 accel defaultsite=[2001:db8::10]
cache_peer 2001:db8::10 parent 80 0 no-query no-digest
           originserver name=ReverseProxy
acl our_sites dst 2001:db8::10
http_access allow our_sites
cache_peer_access ReverseProxy allow our_sites
cache_peer_access ReverseProxy deny all

acl manager proto cache_object
acl localhost src 127.0.0.1/32 ::1
acl to_localhost dst 127.0.0.0/8 0.0.0.0/32 ::1
acl SSL_ports port 443
acl Safe_ports port 80          # http
acl Safe_ports port 21         # ftp

```



```

acl Safe_ports port 443          # https
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535  # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 777         # multiling http
acl CONNECT method CONNECT

http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports
http_access allow localhost
http_access deny all

coredump_dir /var/spool/squid3

refresh_pattern ^ftp:          1440      20% 10080
refresh_pattern ^gopher:      1440      0% 1440
refresh_pattern -i (/cgi-bin/|\?) 0 0% 0
refresh_pattern .              0        20% 4320

```

Seguem-se comentários sobre algumas das diretivas utilizadas.

```
http_port 80 accel defaultsite=[2001:db8::10]
```

Para que o Squid funcione como um *proxy* reverso de forma transparente, o serviço deve escutar requisições na porta 80, de forma semelhante a um servidor *Web*. A diretiva informa, juntamente com a porta 80 a ser utilizada, a configuração de domínio padrão que o *proxy* deve solicitar ao servidor. No caso, o endereço IPv6 de `n1Host` foi configurado, visto que o cenário não utiliza nenhum servidor DNS.

```

cache_peer 2001:db8::10 parent 80 0 no-query no-digest
          originserver name=ReverseProxy

```

Esta diretiva configura o endereço do `n1Host` no *cache* do serviço de *proxy* Squid. As opções `parent 80 0 no-query no-digest originserver` informam que a conexão será realizada com um servidor *Web* em lugar de realizar com outra instância de *cache*. Caso existam mais servidores *Web* na rede interna, eles devem ser configurados de forma semelhante no Squid.

```
acl our_sites dst 2001:db8::10
http_access allow our_sites
cache_peer_access ReverseProxy allow our_sites
cache_peer_access ReverseProxy deny all
```

O conjunto de diretivas apresentado configura o Squid para aceitar apenas os registros cadastrados por meio da lista de controle `our_sites`. Nesta experiência, foi utilizado diretamente o endereço IPv6 de `n1Host`, entretanto usualmente o nome da página *Web* é utilizado diretamente com a opção `dstdomain` no local de `dst`.

O restante das diretivas compõe a configuração padrão indicada para servidores Squid.

5. Ainda no terminal de `n2Proxy`, inicie o serviço Squid:

```
# squid3 -s -f /etc/squid3/squid.conf
```

O resultado do comando é representado pela Figura 2.60.



Figura 2.60: resultado da inicialização do serviço de *proxy* Squid.

6. Ainda no terminal de `n2Proxy`, verifique a escuta da porta 80:

```
# netstat -antup
```

O resultado do comando deve ser similar à Figura 2.61.

As portas UDP listadas são escolhidas aleatoriamente por padrão e são utilizadas com os módulos ICP, HTCP e DNS do Squid.

```

CORE: n2Proxy (console)
root@n2Proxy:/tmp/pycore.55840/n2Proxy.conf# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp6      0      0 :::80                   :::*                     LISTEN
577/(squid)
udp       0      0 0.0.0.0:53033           0.0.0.0:*
577/(squid)
udp6      0      0 :::45319                :::*
577/(squid)
root@n2Proxy:/tmp/pycore.55840/n2Proxy.conf# █

```

Figura 2.61: listagem das portas escutadas em n2Proxy, incluindo a TCP 80 em conexões IPv6.

- Abra um terminal de n3Client com um duplo-clique e verifique a acessibilidade do serviço HTTP de n1Host por meio do *proxy* seja por IPv4 quanto por IPv6:

```

# wget 203.0.113.10
# wget http://[2001:db8:2::10]/

```

O resultado dos comandos é representado pela Figura 2.62.

Para o acesso a uma página *Web*, usualmente é utilizado o nome de domínio, de modo a consultar o registro DNS equivalente. Para tal situação, o registro DNS deve apontar para o endereço do *proxy* no local do endereço IP da página *Web*.

- Abra um terminal de n2Proxy com um duplo-clique e verifique as entradas no *log* do Squid relativas aos acessos a n1Host:

```

# cat /var/log/squid3/access.log

```

O resultado do comando é representado pela Figura 2.63.

```

CORE: n3Client (console)
root@n3Client:/tmp/pycore.55840/n3Client.conf# wget 203.0.113.10
--2013-04-10 20:04:03-- http://203.0.113.10/
Connecting to 203.0.113.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html'

100%[=====] 177      --.-K/s  in 0s

2013-04-10 20:04:03 (29.1 MB/s) - `index.html' saved [177/177]

root@n3Client:/tmp/pycore.55840/n3Client.conf# wget http://[2001:db8:2::10]/
--2013-04-10 20:04:22-- http://[2001:db8:2::10]/
Connecting to 2001:db8:2::10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html,1'

100%[=====] 177      --.-K/s  in 0s

2013-04-10 20:04:22 (29.6 MB/s) - `index.html,1' saved [177/177]

root@n3Client:/tmp/pycore.55840/n3Client.conf# █

```

Figura 2.62: acesso do serviço HTTP oriundo de n3Client por meio do proxy.

```

CORE: n2Proxy (console)
root@n2Proxy:/tmp/pycore.55840/n2Proxy.conf# cat /var/log/squid3/access.log
1365635043.960      6 192.0.2.20 TCP_MISS/200 563 GET http://[2001:db8:10]/ - F
IRST_UP_PARENT/ReverseProxy text/html
1365635062.337      0 2001:db8:1::20 TCP_MEM_HIT/200 570 GET http://[2001:db8:1
0]/ - NONE/- text/html
root@n2Proxy:/tmp/pycore.55840/n2Proxy.conf# █

```

Figura 2.63: verificação do log de acesso do Squid em n2Proxy.

Nos registros constam as requisições efetuadas por n3Client, cujos endereços são 192.0.2.20 e 2001:db8:1::20, a n1Host, cujo endereço é 2001:db8::10, por meio do *proxy*.

O servidor Squid registra tanto endereços IPv4 quanto endereços IPv6. Esse fato é relevante, pois, caso sejam utilizados *scripts* ou outras ferramentas de análise de *log*, tais ferramentas devem ser capazes de reconhecer corretamente o formato de endereços IPv6.

9. Abra um terminal de n1Host com um duplo-clique e verifique as entradas no *log* do Apache relativas aos acessos a n1Host:

```
# cat /var/log/apache2/access.log
```

O resultado do comando deve ser similar à Figura 2.64.



```
CORE: n1Host (console)
root@n1Host:/tmp/pycore.55840/n1Host.conf# cat /var/log/apache2/access.log
2001:db8::11 - - [10/Apr/2013:20:04:03 -0300] "GET / HTTP/1.1" 200 490 "-" "wget
/1.13.4 (linux-gnu)"
root@n1Host:/tmp/pycore.55840/n1Host.conf#
```

Figura 2.64: verificação do log de acesso do Apache em n1Host.

Note que consta somente um acesso a n1Host, uma vez que a segunda requisição foi respondida diretamente usando o *cache* do serviço *proxy*.

A primeira requisição enviada para n3Client pode ser visualizada na linha por meio da expressão “GET / HTTP/1.1”.

Verifique que, como o Squid não foi configurado para atuar como um *proxy* transparente, o Apache registrou a requisição do endereço IPv6 de n2Proxy 2001:db8::11 em lugar do endereço de n3Client.

10. Abra um terminal de n2Proxy e pare o serviço de *proxy* Squid. Execute o seguinte comando:

```
# squid3 -k shutdown
```

O resultado do comando deve ser similar à Figura 2.65.



```
CORE: n2Proxy (console)
root@n2Proxy:/tmp/pycore.55840/n2Proxy.conf# squid3 -k shutdown
root@n2Proxy:/tmp/pycore.55840/n2Proxy.conf#
```

Figura 2.65: resultado esperado da parada do serviço de *proxy* Squid em n2Proxy.

11. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 2.8. Samba: compartilhamento de arquivos em rede IPv6

Objetivo

Esta experiência apresenta a configuração básica de um serviço Samba que disponibilizará o acesso a uma pasta compartilhada a um cliente, utilizando comunicação apenas em IPv6.

Para a realização do presente exercício será utilizada a topologia descrita no arquivo: **2-08-samba.imn**.

Introdução teórica

Originalmente especificado pela Microsoft, IBM e Intel, o *Server Message Block* (SMB) é um protocolo que gerencia o acesso remoto e o compartilhamento de arquivos e recursos de impressão.

Samba é uma implementação dos protocolos SMB/CIFS (*Common Internet File System*) em *software* livre, desenvolvida para sistemas Unix, que apresenta como principal característica o compartilhamento e gerenciamento de recursos de um servidor por clientes que utilizem diferentes sistemas operacionais. O suporte ao IPv6 está presente desde a versão 3.2.0 do Samba, tanto nas ferramentas de servidor quanto nos clientes.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **2-08-samba.imn** localizado no diretório `lab`, dentro do `Desktop`. A topologia de rede, representada pela Figura 2.66, deve aparecer.

O objetivo dessa topologia de rede é representar o mínimo necessário para que o serviço Samba seja analisado.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós `n1Host` e `n2Client` e a conectividade entre eles.

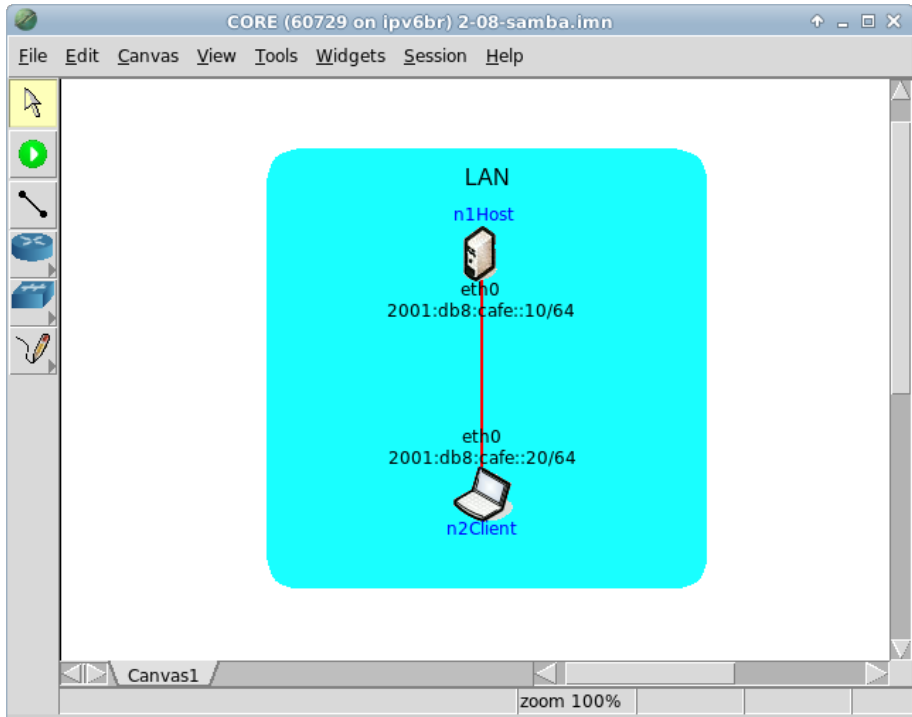


Figura 2.66: *topologia da Experiência 2.8 no CORE.*

3. Abra um terminal de n1Host com um duplo-clique e prepare o sistema antes da configuração do próprio serviço Samba:

```
# useradd -g users -p bar foo
# mkdir /usr/local/share/samba/dir
# touch /usr/local/share/samba/dir/42.txt
# chown -R foo:users /usr/local/share/samba/dir
```

O resultado dos comandos é representado pela Figura 2.67.

Segue-se uma descrição dos quatro comandos utilizados:

- Adiciona ao sistema operacional o usuário `foo` com senha `bar`.
- Cria o diretório `dir`, que será utilizado para armazenar os arquivos do usuário por meio do serviço Samba.
- Aloca um arquivo de texto vazio no diretório criado.
- Transfere a posse do diretório criado ao novo usuário.



```

n1Host
root@n1Host:/tmp/pycore.36862/n1Host.conf# useradd -g users -p bar foo
root@n1Host:/tmp/pycore.36862/n1Host.conf# mkdir /usr/local/share/samba/dir
root@n1Host:/tmp/pycore.36862/n1Host.conf# touch /usr/local/share/samba/dir/42.txt
root@n1Host:/tmp/pycore.36862/n1Host.conf# chown -R foo:users /usr/local/share/samba/dir
root@n1Host:/tmp/pycore.36862/n1Host.conf# █

```

Figura 2.67: resultado da configuração inicial para o usuário foo em n1Host.

4. Configure o servidor Samba de modo a compartilhar um diretório na rede por meio do usuário criado.
- (a) Ainda no terminal de n1Host, edite o arquivo de configuração do Samba para adicionar o trecho em **negrito**, localizado em /etc/samba/smb.conf. O arquivo deve ser composto pelas seguintes linhas:

```

[global]
workgroup = crew
netbios name = node
security = user
interfaces = 2001:db8:cafe::64
hosts allow = 2001:db8:cafe::20
load printers = no
log file = /var/log/samba.%m
max log size = 50
[data]
path = /usr/local/share/samba/dir
read only = yes
valid users = foo

```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano. Verifique o conteúdo do arquivo de configuração do Samba:

```
# cat /etc/samba/smb.conf
```

O resultado do comando é representado pela Figura 2.68.



```

root@n1Host:/tmp/pycore.36862/n1Host.conf# cat /etc/samba/smb.conf
[global]
workgroup = crew
netbios name = node
security = user
interfaces = 2001:db8:cafe::/64
hosts allow = 2001:db8:cafe::20
load printers = no
log file = /var/log/samba.%m
max log size = 50
[data]
path = /usr/local/share/samba/dir
read only = yes
valid users = foo
root@n1Host:/tmp/pycore.36862/n1Host.conf# █

```

Figura 2.68: *verificação do arquivo de configuração do Samba.*

Note que nas diretivas de configuração constam dois endereços IPv6. O parâmetro `interfaces` indica em quais interfaces de rede ou sub-redes o serviço Samba será habilitado, sendo no presente caso por meio da sub-rede `2001:db8:cafe::/64`. O campo `hosts allow` lista os endereços com permissão de acesso ao serviço, sendo neste caso somente o endereço `2001:db8:cafe::20`.

- (b) Ainda no terminal de `n1Host`, valide a configuração editada:

```
# testparm
```

O resultado do comando é representado pela Figura 2.69.

- (c) Ainda no terminal de `n1Host`, configure a senha do usuário `foo` no serviço Samba:

```
# smbpasswd -a foo
```

Quando solicitado, utilize a mesma senha (`bar`) definida anteriormente. O resultado do comando é representado pela Figura 2.70.

Para a devida configuração, é necessário que as senhas do usuário do sistema `foo` e do usuário do Samba `foo` coincidam.

- (d) Ainda no terminal de `n1Host`, inicie o serviço Samba:

```
# smb
```

O resultado do comando é representado pela Figura 2.71.



```

n1Host
root@n1Host:/tmp/pycore.36862/n1Host.conf# testparm
Load smb config files from /etc/samba/smb.conf
rlimit_max: increasing rlimit_max (1024) to minimum Windows limit (16384)
Processing section "[data]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions

[global]
    workgroup = CREW
    netbios name = NODE
    interfaces = 2001:db8:cafe::/64
    log file = /var/log/samba.%m
    max log size = 50
    load printers = No
    idmap config * : backend = tdb
    hosts allow = 2001:db8:cafe::20

[data]
    path = /usr/local/share/samba/dir
    valid users = foo
root@n1Host:/tmp/pycore.36862/n1Host.conf# █

```

Figura 2.69: resultado da validação da configuração do Samba.



```

n1Host
root@n1Host:/tmp/pycore.36862/n1Host.conf# smbpasswd -a foo
New SMB password:
Retype new SMB password:
Added user foo.
root@n1Host:/tmp/pycore.36862/n1Host.conf# █

```

Figura 2.70: resultado da definição de senha do usuário foo para o serviço Samba.



```

n1Host
root@n1Host:/tmp/pycore.36862/n1Host.conf# smbd
root@n1Host:/tmp/pycore.36862/n1Host.conf# █

```

Figura 2.71: resultado da inicialização do serviço de compartilhamento de recursos Samba.

- (e) Ainda no terminal de n1Host, verifique o funcionamento do serviço Samba. Execute o seguinte comando:

```
# ps aux
```

O resultado do comando é representado pela Figura 2.72.

```

root@n1Host:/tmp/pycore.36862/n1Host.conf# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   2296    516 ?        S      16:44   0:00 /usr/sbin/vnode
root        43  0.0  0.1   3492   1404 pts/6    Ss     16:44   0:00 /bin/bash
root        83  0.0  0.4  21396   3276 ?        Ss     16:56   0:00 smbd
root        85  0.0  0.1   21500   1316 ?        S      16:56   0:00 smbd
root        86  0.0  0.1   2864    1040 pts/6    R+     16:56   0:00 ps aux
root@n1Host:/tmp/pycore.36862/n1Host.conf# █

```

Figura 2.72: listagem dos processos do servidor, incluindo smbd.

5. Configure n2Client para acessar o serviço Samba.
 - (a) Abra um terminal de n2Client com um duplo-clique e inicie o serviço smbclient para acesso ao serviço Samba. Para isto execute:

```
# smbclient //2001:db8:cafe::10/data -U foo
```

Quando a senha do usuário foo for solicitada, digite bar, conforme a configuração anterior.

O resultado do comando é representado pela Figura 2.73.

```

root@n2Client:/tmp/pycore.36862/n2Client.conf# smbclient //2001:db8:cafe::10/dat
a -U foo
Enter foo's password:
Domain=[CREW] OS=[Unix] Server=[Samba 3.6.3]
smb: \> █

```

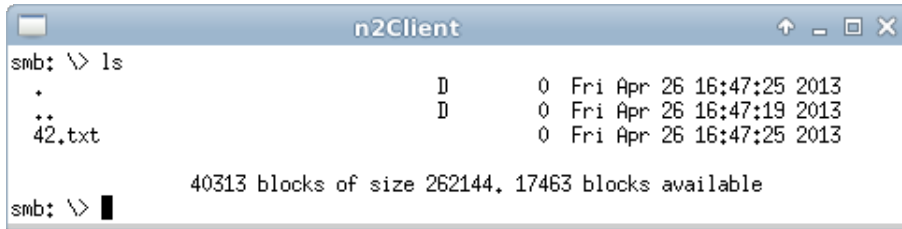
Figura 2.73: resultado da inicialização do cliente de serviço de compartilhamento de recursos smbclient.

Caso deseje mais informações sobre os comandos disponíveis pelo Samba, utilize o comando help.

- (b) Ainda no terminal provido por smbclient, verifique o arquivo alocado no servidor:

```
smb: \> ls
```

O resultado do comando é representado pela Figura 2.74.



```

n2Client
smb: \> ls
.           D           0 Fri Apr 26 16:47:25 2013
..          D           0 Fri Apr 26 16:47:19 2013
42.txt      0 Fri Apr 26 16:47:25 2013

40313 blocks of size 262144, 17463 blocks available
smb: \> █

```

Figura 2.74: resultado da listagem de arquivos compartilhados em n1Host por meio do Samba.

Encerre a execução de smbclient com o comando `exit` ou da combinação de teclas `Ctrl+D`.

- (c) Além do acesso pelo terminal de smbclient, é possível efetuar o mapeamento de um diretório para um *drive* de rede.

Ainda no terminal de n2Client, crie o diretório para o mapeamento. Para isto execute o seguinte comando:

```
# mkdir mnt
```

O resultado do comando é representado pela Figura 2.75.



```

n2Client
root@n2Client:/tmp/pycore.36862/n2Client.conf# mkdir mnt
root@n2Client:/tmp/pycore.36862/n2Client.conf# █

```

Figura 2.75: resultado da alocação de diretório para mapeamento em n2Client.

- (d) Ainda no terminal de n2Client, efetue o mapeamento do Samba. Para isto execute o comando:

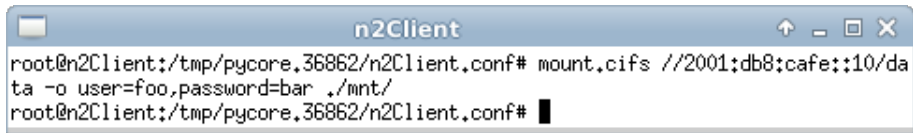
```
# mount.cifs //2001:db8:cafe::10/data -o user=foo,password=bar
./mnt/
```

O resultado do comando é representado pela Figura 2.76.

- (e) Ainda no terminal de n2Client, valide o mapeamento. Execute os seguintes comandos:

```
# cd mnt/
# ls
```

O resultado dos comandos é representado pela Figura 2.77.



```
n2Client
root@n2Client:/tmp/pycore.36862/n2Client.conf# mount.cifs //2001:db8:cafe::10/data
-o user=foo,password=bar ./mnt/
root@n2Client:/tmp/pycore.36862/n2Client.conf# █
```

Figura 2.76: mapeamento do diretório do usuário foo utilizando do Samba em n2Client.



```
n2Client
root@n2Client:/tmp/pycore.36862/n2Client.conf# cd mnt/
root@n2Client:/tmp/pycore.36862/n2Client.conf/mnt# ls
42.txt
root@n2Client:/tmp/pycore.36862/n2Client.conf/mnt# █
```

Figura 2.77: validação do mapeamento do diretório do usuário foo utilizando Samba em n2Client.

- (f) Abra um terminal de n1Host com um duplo-clique e verifique a lista de portas escutadas. Utilize o seguinte comando para isto:

```
# netstat -antup
```

O resultado do comando é representado pela Figura 2.78.



```
n2Client
root@n2Client:/tmp/pycore.36862/n2Client.conf/mnt# netstat -antup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp6      0      0 2001:db8:cafe::20:45469 2001:db8:cafe::10:445  ESTABLISHED
-
root@n2Client:/tmp/pycore.36862/n2Client.conf/mnt# █
```

Figura 2.78: listagem das portas escutadas em n1Host.

Observe a conexão do Samba através da porta 445 no endereço IPv6 de n1Host 2001:db8:cafe::10. Para utilizar o serviço Samba, n2Client utiliza alguma porta disponível entre 1024 e 49151, definida para usuários de serviços, conforme especificado em (IANA, 2014).

6. Prepare-se para encerrar a simulação.
 - (a) Abra um terminal de `n2Client` com um duplo-clique e remova o mapeamento do serviço Samba. Digite o seguinte comando no terminal:

```
# umount mnt/
```

O resultado do comando é representado pela Figura 2.79.



```
n2Client
root@n2Client:/tmp/pycore.36862/n2Client.conf# umount mnt/
root@n2Client:/tmp/pycore.36862/n2Client.conf# █
```

Figura 2.79: resultado da remoção do mapeamento Samba em `n2Client`.

- (b) Abra um terminal de `n1Host` e remova o usuário `foo` do Samba:

```
# smbpasswd -x foo
```

O resultado do comando é representado pela Figura 2.80.



```
n1Host
root@n1Host:/tmp/pycore.36862/n1Host.conf# smbpasswd -x foo
Deleted user foo.
root@n1Host:/tmp/pycore.36862/n1Host.conf# █
```

Figura 2.80: resultado da remoção do usuário `foo` para o serviço Samba.

- (c) Ainda no terminal de `n1Host`, remova o usuário `foo` do sistema:

```
# userdel foo
```

O resultado do comando é representado pela Figura 2.81.



```
n1Host
root@n1Host:/tmp/pycore.36862/n1Host.conf# userdel foo
root@n1Host:/tmp/pycore.36862/n1Host.conf# █
```

Figura 2.81: resultado da remoção do usuário `foo` do sistema da VM.

7. Encerre a simulação, conforme descrito no Apêndice B.

Capítulo 3

Segurança

Experiência 3.1. Ataque DoS ao *Neighbor Discovery Protocol*

Objetivo

Esta experiência tem como objetivo apresentar uma rede local afetada por um ataque de negação de serviço (*Denial of Service* – DoS), baseado no envio de falsas mensagens NA, para que este possa ser analisado. Nesse cenário, a máquina `n3Faker` responderá a todas as requisições NS, informando ser o portador do endereço IPv6 verificado pelo processo de *Duplicate Address Detection* (DAD) e impedindo que novas máquinas obtenham conectividade. Na sequência, o *software* NDPMon será utilizado para demonstrar como este tipo de ataque pode ser detectado.

Para o presente exercício será utilizada a topologia descrita no arquivo: **3-01-DoS-NA.imn**.

Introdução teórica

O protocolo de descoberta de vizinhança, o NDP, foi desenvolvido para automatizar configurações nos dispositivos de uma rede e para facilitar a comunicação entre eles. Entre suas funções está a detecção de dispositivos da rede, a determinação dos endereços físicos dos nós vizinhos, a localização dos roteadores na rede e a detecção do uso de endereços IP duplicados. Todas essas funções atuam baseadas na troca de mensagens ICMPv6.

Assim como ocorre com o protocolo antigo, o IPv4, algumas características do IPv6 podem ser exploradas por usuários mal intencionados para a realização de ataques a outros usuários ou até mesmo a toda uma rede de computadores.

Um exemplo é o uso da funcionalidade DAD para gerar um ataque de negação de serviço a uma rede IPv6. A DAD ocorre quando um novo dispositivo conectado à rede envia uma mensagem NS para verificar se o endereço IPv6 atribuído a sua interface já está sendo utilizado por outro nó. Se o endereço já estiver em uso, o dispositivo que está utilizando esse IP envia uma mensagem NA contendo essa informação e o dispositivo que realizou a requisição fica impedido de utilizar esse endereço.

O ataque de negação de serviço, que explora essa funcionalidade, ocorre quando um computador infectado por um *malware* passa a responder a todas as mensagens NS enviadas na rede, mesmo as que não são endereçadas a ele, informando que ele está utilizando qualquer endereço IPv6 que esteja sendo verificado no processo de DAD. Com isso, todo novo nó que tente se conectar a essa rede não conseguirá obter um endereço válido e estabelecer comunicações.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **3-01-DoS-NA.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 3.1, deve aparecer.

O objetivo dessa topologia de rede é representar o mínimo necessário para que o ataque seja analisado.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação e verifique a configuração de endereços IPv6 nos nós n1Host, n2Host e n3Faker.

Ao contrário de outras experiências, não atribuímos endereços *unicast* globais às interfaces de rede, pois o processo será realizado utilizando somente os endereços *unicast link-local*. O processo de DAD é realizado para toda atribuição de endereços IPv6.

3. Abra um terminal de n1Host, com um duplo-clique e execute um ping6 para o n2Host:

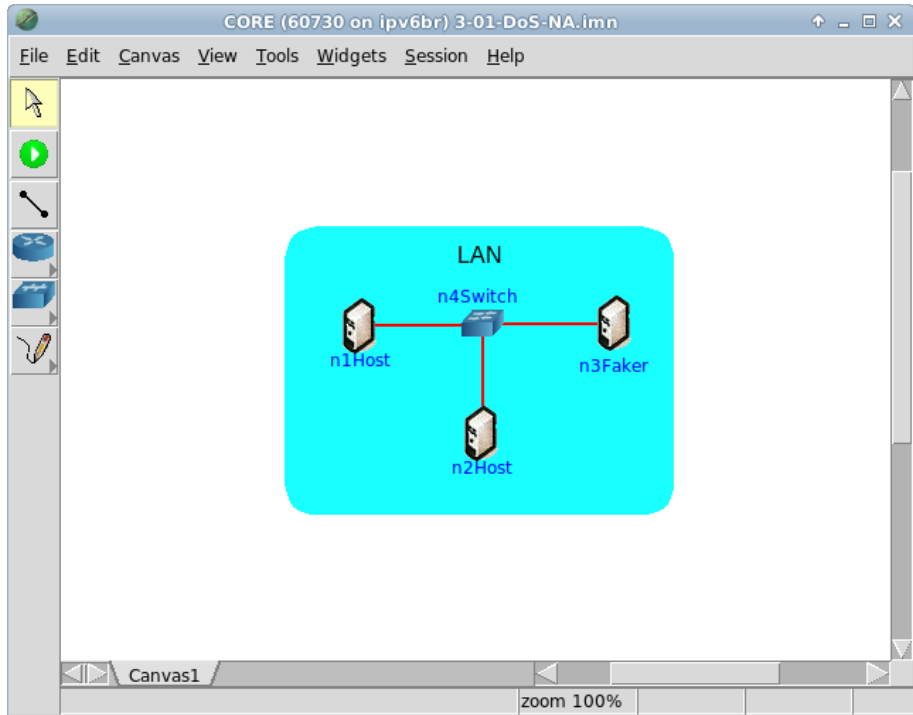


Figura 3.1: *topologia da Experiência 3.1 no CORE.*

```
# ping6 -c 4 -I eth0 fe80::200:ff:feaa:1
```

O resultado do comando é representado pela Figura 3.2.

```
n1Host
root@n1Host:/tmp/pycore.33741/n1Host.conf# ping6 -c 4 -I eth0 fe80::200:ff:feaa:1
1
PING fe80::200:ff:feaa:1(fe80::200:ff:feaa:1) from fe80::200:ff:feaa:0 eth0: 56
data bytes
64 bytes from fe80::200:ff:feaa:1: icmp_seq=1 ttl=64 time=0,185 ms
64 bytes from fe80::200:ff:feaa:1: icmp_seq=2 ttl=64 time=0,240 ms
64 bytes from fe80::200:ff:feaa:1: icmp_seq=3 ttl=64 time=0,056 ms
64 bytes from fe80::200:ff:feaa:1: icmp_seq=4 ttl=64 time=0,177 ms

--- fe80::200:ff:feaa:1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0,056/0,164/0,240/0,068 ms
root@n1Host:/tmp/pycore.33741/n1Host.conf# █
```

Figura 3.2: *verificação da conectividade com n2Host.*

4. Em paralelo, efetue:

- (a) A coleta dos pacotes trafegados na interface `eth0` de `n3Faker`. As instruções de coleta de pacotes utilizando `tcpdump` ou `Wireshark` se encontram no Apêndice C.
- (b) Efetue os seguintes passos enquanto a coleta é realizada:
- i. Abra outro terminal de `n3Faker` com um duplo-clique e utilize o seguinte comando para realizar o ataque:

```
# dos-new-ip6 eth0
```

O `dos-new-ip6` é um executável que pertence ao pacote `THC-IPv6`. Este pacote é uma ferramenta desenvolvida para detectar vulnerabilidades de segurança do IPv6 e do ICMPv6. O `dos-new-ip6` atua escutando todos os endereços *multicast solicited-node* e respondendo as consultas do DAD informando que todos os endereços testados estão em uso, gerando um ataque de DoS.

- ii. Abra um terminal de `n1Host`, desative a interface de rede `eth0` e ative-a para verificar a negação de serviço:

```
# ip link set eth0 down
# ip link set eth0 up
# ip addr show eth0
# ifconfig eth0
```

O resultado dos comandos é representado pela Figura 3.3.



```
n1Host
root@n1Host:/tmp/pycore.33741/n1Host.conf# ip link set eth0 down
root@n1Host:/tmp/pycore.33741/n1Host.conf# ip link set eth0 up
root@n1Host:/tmp/pycore.33741/n1Host.conf# ip addr show eth0
107: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether 00:00:00:aa:00:00 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::200:ff:feaa:0/64 scope link tentative dadfailed
        valid_lft forever preferred_lft forever
root@n1Host:/tmp/pycore.33741/n1Host.conf# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:00:00:aa:00:00
          inet6 addr: fe80::200:ff:feaa:0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:51 errors:0 dropped:0 overruns:0 frame:0
          TX packets:15 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8732 (8.7 KB)  TX bytes:1362 (1.3 KB)

root@n1Host:/tmp/pycore.33741/n1Host.conf# █
```

Figura 3.3: resultado do processo DAD em `n1Host`.

Mediante esta sequência de comandos, a interface `eth0` efetua o procedimento de DAD para a obtenção de endereço IPv6 de *link-local*. Note que o resultado do comando `ip addr show eth0` indica que não houve a atribuição do endereço por meio do texto `tentative dadfailed`. Em paralelo, o comando `ifconfig eth0` não indicou qualquer tipo de falha durante a atribuição de endereço IPv6.

O resultado em `n3Faker` da execução de `dos-new-ip6` é representado pela Figura 3.4.



```

root@n3Faker:/tmp/pycore.33741/n3Faker.conf# dos-new-ip6 eth0
Started ICMP6 DAD Denial-of-Service (Press Control-C to end) ...
Spoofed packet for existing ip6 as fe80::200:ff:feaa:0

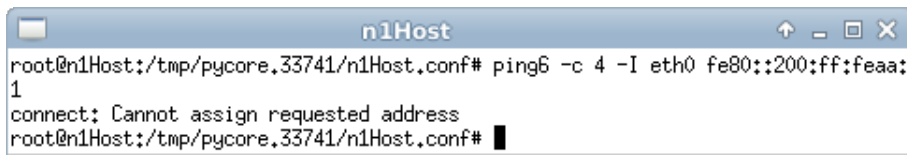
```

Figura 3.4: resultado do processo DAD em `n3Faker`.

- iii. Ainda no terminal de `n1Host`, verifique novamente a conectividade com `n2Host`:

```
# ping6 -c 4 -I eth0 fe80::200:ff:feaa:1
```

O resultado do comando é representado pela Figura 3.5.



```

root@n1Host:/tmp/pycore.33741/n1Host.conf# ping6 -c 4 -I eth0 fe80::200:ff:feaa:1
1
connect: Cannot assign requested address
root@n1Host:/tmp/pycore.33741/n1Host.conf#

```

Figura 3.5: verificação do processo DAD em `n1Host`.

Note que tal comando confirma que o endereço *unicast link-local* não foi atribuído à interface `eth0` de `n1Host`.

- iv. No terminal de `n3Faker`, encerre o comando `dos-new-ip6` por meio da sequência de teclado `Ctrl+C`.
5. Efetue a análise dos pacotes coletados na interface `eth0` de `n3Faker`. Aplique o filtro `icmpv6` no Wireshark e procure pelos pacotes NS e NA.

O pacote NS enviado por `n1Host` foi recebido por `n3Faker`. Como resposta à mensagem, `n3Faker` enviou pacotes NA ao `n1Host` para informar que o endereço IPv6 *unicast link-local* `fe80::200:ff:feaa:0` já está sendo utilizado, apesar do endereço IPv6 *unicast link-local* de `n3Faker` ser `fe80::200:ff:feaa:2`, conforme verificado no passo 2.

O comportamento de `n3Faker`, por meio do comando `dos-new-ip6`, gera uma negação de serviço, pois impede que qualquer dispositivo que venha a pertencer ao enlace consiga ativar sua interface de comunicação.

Este ataque poderia ser evitado com a utilização da suíte de protocolos SEND (*Secure Neighbor Discovery*), descrito na RFC 3971 (Arkko *et al.*, 2005). Entretanto, durante o desenvolvimento desta experiência, ainda não existia uma implementação funcional para Linux.

Nos próximos passos, o `NDPMon`, uma ferramenta para a geração de *logs* e registros de comportamentos suspeitos no enlace, será utilizada para auxiliar na detecção de ataques e na identificação de máquinas atacantes.

6. Encerre a simulação, conforme descrito no Apêndice B, e inicialize novamente a simulação.
7. Abra um terminal de `n3Faker`, com um duplo-clique e efetue o ataque de negação de serviço. No terminal execute o comando:

```
# dos-new-ip6 eth0
```

8. Abra um terminal de `n2Host`, com um duplo-clique, e inicie o monitoramento por meio do `NDPMon`. Para isto o seguinte comando deve ser executado:

```
# ndpmon -i eth0 -v
```

9. Abra um terminal de `n1Host`, desative a interface de rede `eth0` e ative-a para verificar a negação de serviço:

```
# ip link set eth0 down  
# ip link set eth0 up  
# ip addr show eth0
```

Após a execução deste procedimento, analise o resultado do monitoramento do enlace realizado em `n2Host`. O *log* gerado pelo NDPMon indica, por meio da linha `[monitoring_na] New Ethernet DAD DoS`, que ocorreu um ataque de negação de serviço no processo de detecção de endereço duplicado. Também é possível analisar qual endereço sofreu a negação de serviço verificando o campo `Target Address` da mensagem NA capturada pelo NDPMon.

A Figura 3.6 representa o *log* gerado pelo NDPMon. Note que esta figura apresenta apenas parte das informações geradas, visto que seu conteúdo é muito extenso.

O NDPMon também pode ser configurado para alertar em caso de detecção de ataques. Por exemplo, por meio do envio de *e-mails*, permitindo uma rápida ação dos administradores de rede para mitigar os problemas causados.

```

n2Host
root@n2Host:/tmp/pycore.33924/n2Host.conf# ndpmon -i eth0
----- Initialization -----
interface: eth0
Reading configuration file: "/usr/local/etc/ndpmon/config_ndpmon.xml" ...
[settings] NDPMon general settings: {
  actions high priority {
    syslog
    no sendmail
    no pipe program
  }
  actions low priority {
    syslog
    no sendmail
    no pipe program
  }
  admin mail root@localhost
  ignor autoconf
  syslog facility LOG_LOCAL1
  no use reverse hostlookups
}
[parser] Finished reading the configuration.
Reading neighbors file: "/var/local/lib/ndpmon/neighbor_list.xml" ...
[parser] Finished reading the neighbor cache.
-----

[capture_pcap] Listening on interface eth0.
----- ND_NEIGHBOR_SOLICIT -----
Setting LAST DAD ADDR
-----

----- ND_NEIGHBOR_ADVERT -----
[monitoring_na] New Ethernet DAD DoS
[alerts] Alert "dad dos" raised on probe "eth0".
-----

----- ND_NEIGHBOR_ADVERT -----
[monitoring_na] New Ethernet DAD DoS
[alerts] Alert "dad dos" raised on probe "eth0".
-----

```

Figura 3.6: resultado do monitoramento NDPMon em n2Host.

10. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 3.2. *Firewall stateful*

Objetivo

Esta experiência tem como objetivo de ensinar boas práticas para a configuração de *firewalls* IPv6, evitando que configurações incorretas atrapalhem o funcionamento do IP. Para tanto, serão seguidas as recomendações da RFC 4890 (Davies e Mohacsi, 2007). Também serão alteradas algumas regras deste *firewall* para entender as consequências de uma configuração de *firewall* incorreta no funcionamento do IPv6. Este exemplo não faz qualquer tipo de filtro por destinos ou tipos de pacotes que não sejam ICMPv6 e tem por objetivo ser a base para a criação de um *firewall* que atenda às necessidades da maioria das redes.

Para o presente exercício será utilizada a topologia descrita no arquivo: **3-02-firewall-stateful.imn**.

Introdução teórica

Firewalls são equipamentos de rede ou programas de computador que por meio do bloqueio seletivo de conexões, baseados em uma política de segurança, buscam proteger redes de computadores.

Existem dois tipos básicos de *firewall*:

Firewall stateful

É um tipo de *firewall* que guarda o estado das conexões ativas, permitindo apenas a passagem de pacotes que pertençam a essas conexões. Por exemplo, um *firewall* configurado na rede A permitirá, sem restrições, a passagem de pacotes no sentido da rede A para a rede B. Mas pacotes da rede B para a rede A somente passarão pelo *firewall* se forem respostas a solicitações feitas pela rede A, descartando os demais pacotes neste sentido.

Firewall stateless

É um tipo de *firewall* que não guarda o estado das conexões ativas e a decisão sobre um pacote poder ou não passar pelo *firewall* é tomada com base em algumas regras. Estas regras podem ser restritivas ou

permissivas. Um *firewall stateless* com regras restritivas bloqueia a passagem de todos os pacotes que estão definidos nas regras. Se o pacote não tiver uma proibição explícita ele passa pelo *firewall*. Um *firewall stateless* com regras permissivas tem o comportamento oposto, em que todos os pacotes são bloqueados e somente aqueles que possuem regras permissivas passam pelo *firewall*.

No *firewall* IPv4, é bastante comum haver uma política de bloquear todos os pacotes ICMPv4. Entretanto esta política não é compatível com o IPv6. O ICMPv6 teve como base o protocolo ICMPv4, porém o ICMPv6 também executa funções que eram exercidas por outros protocolos no IPv4, como o ARP, RARP e IGMP. Esta mudança foi feita com o objetivo de reduzir a quantidade de protocolos utilizados e, assim, aumentar a coerência e facilitar as implementações. No IPv4, os pacotes de ARP, RARP e IGMP não são filtrados por *firewall*, pois caso fossem, a descoberta de vizinhança não seria possível e as redes não funcionariam. Desta maneira, os mecanismos equivalentes presentes no ICMPv6 não podem ser bloqueados.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **3-02-firewall-stateful.imn** localizado no diretório `lab`, dentro do `Desktop`. A topologia de rede, representada pela Figura 3.7, deve aparecer.

O objetivo da topologia é representar a estrutura mínima necessária para simular um sistema de *firewall*. Essa experiência utiliza uma rede interna composta por `n1Router`, `n3Router`, `n4Host` e `n5Client` e está dividida em duas partes: a configuração de *firewall* em um servidor ou *desktop*, seguido da configuração de um *firewall* em um roteador.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 em todos os nós e a conectividade entre eles.
3. Configure o *firewall* em `n4Host`.

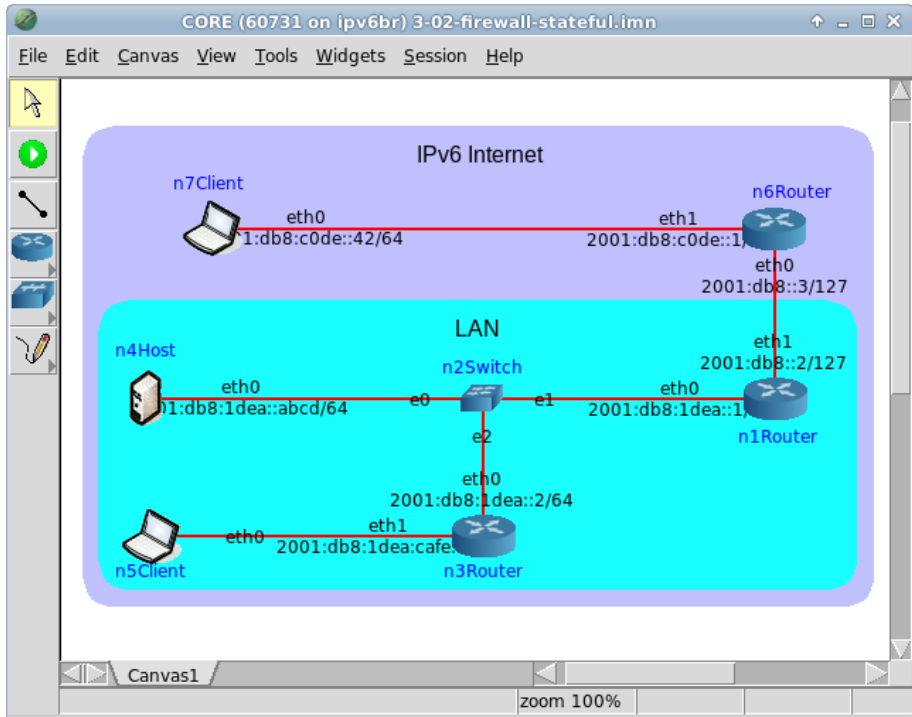


Figura 3.7: topologia da Experiência 3.2 no CORE.

- (a) Abra um terminal de n4Host com um duplo-clique e verifique o conteúdo do arquivo `firewall.sh` por meio do comando:

```
# less firewall.sh
```

O arquivo `firewall.sh` deverá conter as linhas:

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

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
IPTABLES='which ip6tables'
```

```
GLOBAL='2001:db8:1dea::abcd/128'
```

```
INTERNET='2000::/3'
```

```
start() {
```

```

${IPTABLES} -X
${IPTABLES} -P INPUT DROP
${IPTABLES} -F INPUT
${IPTABLES} -P OUTPUT DROP
${IPTABLES} -F OUTPUT
${IPTABLES} -P FORWARD DROP
${IPTABLES} -F FORWARD

for CHAIN in INPUT OUTPUT FORWARD; do
    # Disable Routing Header Type 0 [RFC5095]
    ${IPTABLES} -A ${CHAIN} -m rt --rt-type 0 -j DROP
    # Disable transit for some prefixes
    # 6bone          [RFC5156] : 3ffe::/16
    # BMWG          [RFC5180] : 2001:2::/48
    # ORCHID        [RFC4843] : 2001:10::/28
    # Documentation [RFC3849] : 2001:db8::/32
    # in real life you need to add doc prefix below
    for TARGET in '-s' '-d'; do
        for BLOCKED in '3ffe::/16' '2001:2::/48' '2001:10::/28'; do
            ${IPTABLES} -A ${CHAIN} ${TARGET} ${BLOCKED} -j DROP
        done
    done
done

# Accept All Nodes link-local scope multicast prefix [RFC4291]
${IPTABLES} -A INPUT -d ff02::1 -j ACCEPT

```

```

# Accept Solicited Node link-local scope
# multicast prefix [RFC4291]
## Modify the line below -- 1 of 3 ##
${IPTABLES} -A INPUT -d ff02::1:ff00:0000/104 -j DROP

# Accept local scope ICMPv6 packets from
# link-local prefix [RFC4890]
for IP in 'fe80::/64'; do
    # Destination Unreachable [All codes] (Type 1)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        destination-unreachable -d ${IP} -j ACCEPT
    # Packet Too Big (Type 2)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        packet-too-big -d ${IP} -j ACCEPT
    # Time Exceeded [Code 0] (Type 3)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        ttl-zero-during-transit -d ${IP} -j ACCEPT
    # Time Exceeded [Code 1] (Type 3)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        ttl-zero-during-reassembly -d ${IP} -j ACCEPT
    # Parameter Problem [Code 0] (Type 4)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        bad-header -d ${IP} -j ACCEPT
    # Parameter Problem [Code 1] (Type 4)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        unknown-header-type -d ${IP} -j ACCEPT
    # Parameter Problem [Code 2] (Type 4)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        unknown-option -d ${IP} -j ACCEPT

    # Echo Request (Type 128)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        echo-request -d ${IP} -j ACCEPT
    # Echo Response (Type 129)

```

```

${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    echo-reply -d ${IP} -j ACCEPT

# Router Solicitation (Type 133)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 133 \
    -d ${IP} -j ACCEPT
# Router Advertisement (Type 134)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 134 \
    -d ${IP} -j ACCEPT
# Neighbor Solicitation (Type 135)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 135 \
    -d ${IP} -j ACCEPT
# Neighbor Advertisement (Type 136)
## Modify the line below -- 2 of 3 ##
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 136 \
    -d ${IP} -j DROP
# Inverse Neighbor Discovery Solicitation (Type 141)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 141 \
    -d ${IP} -j ACCEPT
# Inverse Neighbor Discovery Advertisement (Type 142)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 142 \
    -d ${IP} -j ACCEPT

# Listener Query (Type 130)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 130 \
    -d ${IP} -j ACCEPT
# Listener Report (Type 131)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 131 \
    -d ${IP} -j ACCEPT
# Listener Done (Type 132)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 132 \
    -d ${IP} -j ACCEPT

```

```

# Listener Report v2 (Type 143)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 143 \
    -d ${IP} -j ACCEPT

# Certification Path Solicitation (Type 148)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 148 \
    -d ${IP} -j ACCEPT

# Certification Path Advertisement (Type 149)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 149 \
    -d ${IP} -j ACCEPT

# Multicast Router Advertisement (Type 151)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 151 \
    -d ${IP} -j ACCEPT

# Multicast Router Solicitation (Type 152)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 152 \
    -d ${IP} -j ACCEPT

# Multicast Router Termination (Type 153)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 153 \
    -d ${IP} -j ACCEPT

done

for IP in ${GLOBAL}; do
    for ALLOCATED in ${INTERNET}; do
        # Accept routable ICMPv6 packets from
        # allocated prefixes [RFC4890]
        # Destination Unreachable [All codes] (Type 1)
        ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
            destination-unreachable -s ${ALLOCATED} \
            -d ${IP} -j ACCEPT
    done
done

```

```
# Packet Too Big (Type 2)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    packet-too-big -s ${ALLOCATED} \
    -d ${IP} -j ACCEPT

# Time Exceeded [Code 0] (Type 3)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    ttl-zero-during-transit -s ${ALLOCATED} \
    -d ${IP} -j ACCEPT

# Time Exceeded [Code 1] (Type 3)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    ttl-zero-during-reassembly -s ${ALLOCATED} \
    -d ${IP} -j ACCEPT

# Parameter Problem [Code 0] (Type 4)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    bad-header -s ${ALLOCATED} -d ${IP} -j ACCEPT

# Parameter Problem [Code 1] (Type 4)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    unknown-header-type -s ${ALLOCATED} \
    -d ${IP} -j ACCEPT

# Parameter Problem [Code 2] (Type 4)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    unknown-option -s ${ALLOCATED} \
    -d ${IP} -j ACCEPT

# Echo Request (Type 128)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    echo-request -s ${ALLOCATED} -d ${IP} -j ACCEPT

# Echo Response (Type 129)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    echo-reply -s ${ALLOCATED} -d ${IP} -j ACCEPT
```

```
# Router Solicitation (Type 133)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 133 \
    -d ${IP} -j ACCEPT
# Router Advertisement (Type 134)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 134 \
    -d ${IP} -j ACCEPT
# Neighbor Solicitation (Type 135)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 135 \
    -d ${IP} -j ACCEPT
# Neighbor Advertisement (Type 136)
## Modify the line below -- 3 of 3 ##
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 136 \
    -d ${IP} -j DROP
# Inverse Neighbor Discovery Solicitation (Type 141)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 141 \
    -d ${IP} -j ACCEPT
# Inverse Neighbor Discovery Advertisement (Type 142)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 142 \
    -d ${IP} -j ACCEPT

# Listener Query (Type 130)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 130 \
    -d ${IP} -j ACCEPT
# Listener Report (Type 131)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 131 \
    -d ${IP} -j ACCEPT
# Listener Done (Type 132)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 132 \
    -d ${IP} -j ACCEPT
# Listener Report v2 (Type 143)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 143 \
    -d ${IP} -j ACCEPT

# Certification Path Solicitation (Type 148)
```

```

${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 148 \
    -d ${IP} -j ACCEPT
# Certification Path Advertisement (Type 149)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 149 \
    -d ${IP} -j ACCEPT

# Multicast Router Advertisement (Type 151)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 151 \
    -d ${IP} -j ACCEPT
# Multicast Router Solicitation (Type 152)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 152 \
    -d ${IP} -j ACCEPT
# Multicast Router Termination (Type 153)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 153 \
    -d ${IP} -j ACCEPT

# traceroute
${IPTABLES} -A INPUT -p udp -m udp -s ${ALLOCATED} \
    -d ${IP} --dport 33434:33523 -m state \
    --state NEW -j REJECT --reject-with \
    icmp6-port-unreachable

# ssh
${IPTABLES} -A INPUT -p tcp -s ${ALLOCATED} \
    -d ${IP} --dport 22 -j ACCEPT

# http
${IPTABLES} -A INPUT -p tcp -s ${ALLOCATED} \
    -d ${IP} --dport 80 -j ACCEPT

done
done
```



```

# Accept incoming packets from allocated prefixes and
# connection state is ESTABLISHED or RELATED
for ALLOCATED in ${INTERNET}; do
    ${IPTABLES} -A INPUT -s ${ALLOCATED} -m state \
        --state RELATED,ESTABLISHED -j ACCEPT
done

# Send packets from ::/128, link-local and
# global unicast addresses
for IP in '::/128' 'fe80::/64' ${GLOBAL}; do
    ${IPTABLES} -A OUTPUT -s ${IP} -j ACCEPT
done

}

stop () {
    echo "Cleaning 'basename ${IPTABLES}' rules."
    ${IPTABLES} -P INPUT ACCEPT
    ${IPTABLES} -F INPUT
    ${IPTABLES} -P OUTPUT ACCEPT
    ${IPTABLES} -F OUTPUT
    ${IPTABLES} -P FORWARD ACCEPT
    ${IPTABLES} -F FORWARD
}

status () {
    ${IPTABLES} --list -v
}

case "${1}" in
    start)
        start
        ;;
    stop)

```

```
    stop
    ;;
try|test)
    start
    sleep 10
    stop
    ;;
restart|reload|force-reload)
    stop
    sleep 2
    start
    ;;
status)
    status
    ;;
*)
    echo "Usage: ${0} {start|stop|restart|status|try}" >&2
    exit 1
    ;;
esac

exit 0
```

Este *script* foi escrito com base na RFC 4890 (Davies e Mohacsi, 2007). Os três trechos destacados em negrito são regras que derrubam (DROP) mensagens ICMPv6 relacionadas a NS e NA. Nos passos seguintes, será verificado o funcionamento do *firewall* quando essas mensagens são bloqueadas.

- (b) Inicialize no terminal de n4Host o serviço de *firewall* com o comando:

```
# ./firewall.sh start
```

O resultado do comando é representado pela Figura 3.8.



```

n4Host
root@n4Host:/tmp/pycore.55586/n4Host.conf# ./firewall.sh start
root@n4Host:/tmp/pycore.55586/n4Host.conf# █

```

Figura 3.8: *inicialização do serviço de firewall em n4Host.*

- (c) Verifique a conectividade entre n4Host e n7Client.

Ainda no terminal de n4Host, execute os seguintes comandos:

```

# ip -6 neigh flush dev eth0
# ping6 -c 4 2001:db8:c0de::42

```

O resultado dos comandos é representado pela Figura 3.9.

Observe que não existe conectividade entre n4Host e n7Client. A seguir, será verificada a causa desse fato.

- (d) Ainda no terminal de n4Host, execute o comando:

```

# ip -6 neigh show

```

O resultado do comando é representado pela Figura 3.10.



```

n4Host
root@n4Host:/tmp/pycore.55586/n4Host.conf# ip -6 neigh flush dev eth0
root@n4Host:/tmp/pycore.55586/n4Host.conf# ping6 -c 4 2001:db8:c0de::42
PING 2001:db8:c0de::42(2001:db8:c0de::42) 56 data bytes
From 2001:db8:1dea::abcd icmp_seq=1 Destination unreachable: Address unreachable
From 2001:db8:1dea::abcd icmp_seq=2 Destination unreachable: Address unreachable
From 2001:db8:1dea::abcd icmp_seq=3 Destination unreachable: Address unreachable
From 2001:db8:1dea::abcd icmp_seq=4 Destination unreachable: Address unreachable

--- 2001:db8:c0de::42 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3015ms

root@n4Host:/tmp/pycore.55586/n4Host.conf# █

```

Figura 3.9: *verificação de conectividade entre n4Host e n7Client.*



```

n4Host
root@n4Host:/tmp/pycore.55586/n4Host.conf# ip -6 neigh show
fe80::200:ff:feaa:5 dev eth0 lladdr 00:00:00:aa:00:05 DELAY
2001:db8:1dea::1 dev eth0 FAILED
root@n4Host:/tmp/pycore.55586/n4Host.conf# █

```

Figura 3.10: *Verificação de vizinhança em n4Host.*

Apesar de n4Host estar diretamente conectado ao roteador, não foi possível estabelecer uma conexão IPv6, pois n4Host bloqueou as mensagens ICMPv6 NA, que informam os endereços MAC das máquinas do enlace. Note que ao contrário da prática usual de bloquear mensagens ICMP em IPv4, seu bloqueio em IPv6 impossibilita o funcionamento do protocolo, dado seu papel no mapeamento de endereços das camadas de rede e de enlace.

- (e) Ainda no terminal de n4Host, edite o arquivo `firewall.sh` de modo a inserir o trecho em **negrito** e remover o trecho **neste formato**, alterando as três regras destacadas no passo 3.

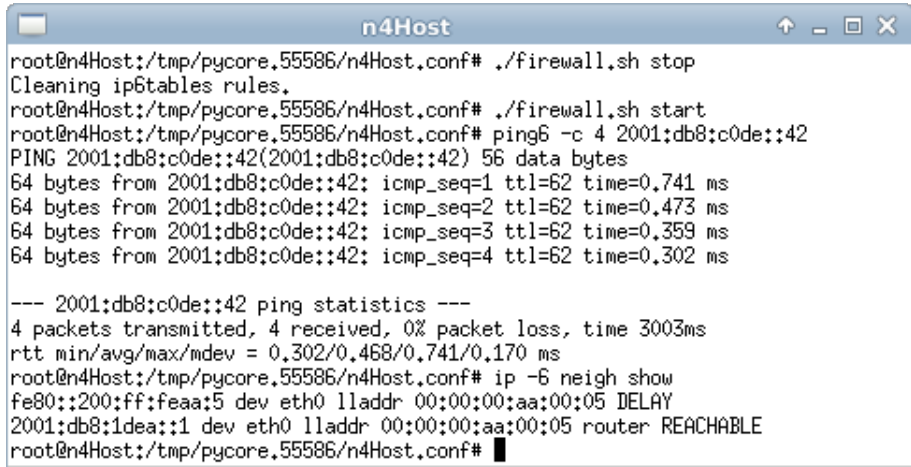
```
...
# Accept Solicited Node link-local scope
# multicast prefix [RFC4291]
## Modify the line below -- 1 of 3 ##
${IPTABLES} -A INPUT -d ff02::1:ff00:0000/104 -j DROP ACCEPT
...
# Neighbor Advertisement (Type 136)
## Modify the line below -- 2 of 3 ##
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 136 \
    -d ${IP} -j DROP ACCEPT
# Inverse Neighbor Discovery Solicitation (Type 141)
...
# Neighbor Advertisement (Type 136)
## Modify the line below -- 3 of 3 ##
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 136 \
    -d ${IP} -j DROP ACCEPT
# Inverse Neighbor Discovery Solicitation (Type 141)
...
```

No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano.

- (f) Ainda no terminal de n4Host, execute os seguintes comandos:

```
# ./firewall.sh stop
# ./firewall.sh start
# ping6 -c 4 2001:db8:c0de::42
# ip -6 neigh show
```

O resultado dos comandos é representado pela Figura 3.11.



```
n4Host
root@n4Host:/tmp/pycore.55586/n4Host.conf# ./firewall.sh stop
Cleaning ip6tables rules.
root@n4Host:/tmp/pycore.55586/n4Host.conf# ./firewall.sh start
root@n4Host:/tmp/pycore.55586/n4Host.conf# ping6 -c 4 2001:db8:c0de::42
PING 2001:db8:c0de::42(2001:db8:c0de::42) 56 data bytes
64 bytes from 2001:db8:c0de::42: icmp_seq=1 ttl=62 time=0,741 ms
64 bytes from 2001:db8:c0de::42: icmp_seq=2 ttl=62 time=0,473 ms
64 bytes from 2001:db8:c0de::42: icmp_seq=3 ttl=62 time=0,359 ms
64 bytes from 2001:db8:c0de::42: icmp_seq=4 ttl=62 time=0,302 ms

--- 2001:db8:c0de::42 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0,302/0,468/0,741/0,170 ms
root@n4Host:/tmp/pycore.55586/n4Host.conf# ip -6 neigh show
fe80::200:ff:feaa:5 dev eth0 lladdr 00:00:00:aa:00:05 DELAY
2001:db8:1dea::1 dev eth0 lladdr 00:00:00:aa:00:05 router REACHABLE
root@n4Host:/tmp/pycore.55586/n4Host.conf# █
```

Figura 3.11: reiniciando o serviço de firewall em n4Host.

Observe que ping6 funcionou corretamente. Como exercício adicional, estude as regras habilitadas no *firewall* e verifique que n4Host está apto a receber somente algumas mensagens relacionadas ao ICMPv6, traceroute6, SSH e HTTP. Referente à sintaxe do *script*, busque informações relacionadas a ip6tables. Quanto as regras utilizadas, verifique a RFC referenciada na introdução teórica desta experiência.

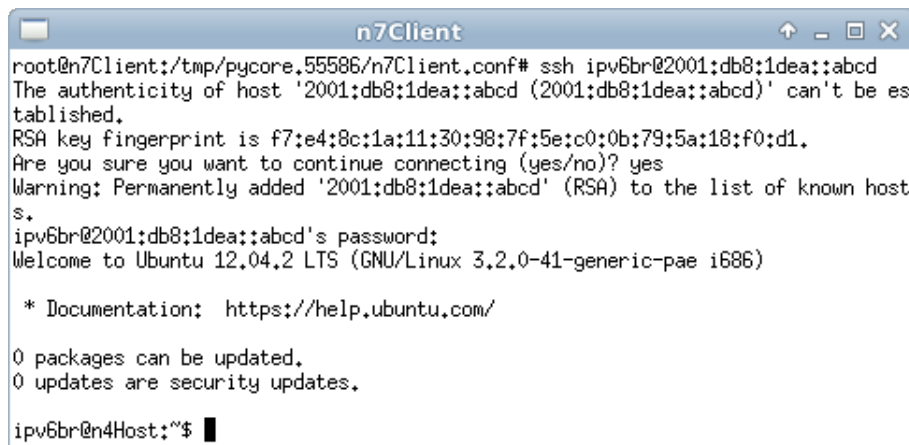
- (g) Abra um terminal de n7Client com um duplo-clique e verifique o acesso ao serviço SSH de n4Host:

```
# ssh ipv6br@2001:db8:1dea::abcd
```

Como é o primeiro acesso de n7Client a n4Host no serviço SSH, será solicitado a inclusão da chave pública RSA de n4Host. Aceite-a respondendo *yes* no terminal. Quando solicitada, a senha de acesso é *ipv6br*.

O resultado do comando é representado pela Figura 3.12.

Após verificar a conectividade por SSH, encerre a sessão com o comando:



```

root@n7Client:/tmp/pycore.55586/n7Client.conf# ssh ipv6br@2001:db8:1dea::abcd
The authenticity of host '2001:db8:1dea::abcd (2001:db8:1dea::abcd)' can't be es
tablished.
RSA key fingerprint is f7:e4:8c:1a:11:30:98:7f:5e:c0:0b:79:5a:18:f0:d1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '2001:db8:1dea::abcd' (RSA) to the list of known host
s.
ipv6br@2001:db8:1dea::abcd's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.2.0-41-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/

0 packages can be updated.
0 updates are security updates.

ipv6br@n4Host:~$ █

```

Figura 3.12: acesso por SSH a n4Host oriundo de n7Client.

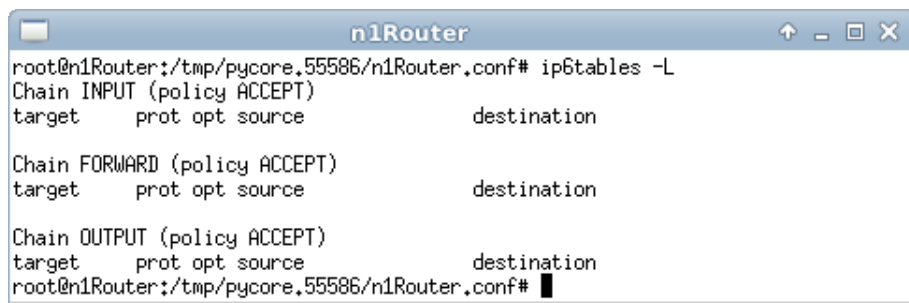
```
# exit
```

4. Configure o *firewall* em n1Router.

- (a) Abra um terminal de n1Router com um duplo-clique e verifique as regras iniciais do *firewall*. Utilize o seguinte comando:

```
# ip6tables -L
```

O resultado do comando é representado pela Figura 3.13.



```

root@n1Router:/tmp/pycore.55586/n1Router.conf# ip6tables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination

Chain FORWARD (policy ACCEPT)
target    prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
root@n1Router:/tmp/pycore.55586/n1Router.conf# █

```

Figura 3.13: verificação das regras iniciais de firewall em n1Router.

Não há nenhum tipo de restrição, uma vez que as políticas de recebimento (*INPUT*), envio (*OUTPUT*) e encaminhamento (*FORWARD*) estão configuradas para aceite (*ACCEPT*).

- (b) Ainda no terminal de n1Router, verifique o conteúdo do arquivo *firewall.sh*:

```
# less firewall.sh
```

O arquivo firewall.sh deverá conter as linhas:

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

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
IPTABLES='which ip6tables'
```

```
GLOBAL='2001:db8:1dea::1/128 2001:db8::2/128'
```

```
LAN='2001:db8:1dea::/48 2001:db8::2/127'
```

```
# IANA aggregated prefixes, according to bcp.nic.br
```

```
IANA_PREFIXES="2800::/12 2001::/32 2001::/16 2600::/12 2610::/23  
                2620::/23 2003::/16 2a00::/12 2400::/12 2c00::/12  
                2002::/16"
```

```
start() {
```

```
    ${IPTABLES} -X
```

```
    ${IPTABLES} -P INPUT DROP
```

```
    ${IPTABLES} -F INPUT
```

```
    ${IPTABLES} -P OUTPUT DROP
```

```
    ${IPTABLES} -F OUTPUT
```

```
    ${IPTABLES} -P FORWARD DROP
```

```
    ${IPTABLES} -F FORWARD
```

```
for CHAIN in INPUT OUTPUT FORWARD; do
```

```
    # Disable Routing Header Type 0 [RFC5095]
```

```
    ${IPTABLES} -A ${CHAIN} -m rt --rt-type 0 -j DROP
```

```
    # Disable transit for some prefixes
```

```
    # 6bone [RFC5156] : 3ffe::/16
```

```
    # BMWG [RFC5180] : 2001:2::/48
```

```
    # ORCHID [RFC4843] : 2001:10::/28
```

```
    # Documentation [RFC3849] : 2001:db8::/32
```

```
    # in real life you need to add doc prefix below
```

```

for TARGET in '-s' '-d'; do
    for BLOCKED in '3ffe::/16' '2001:2::/48' '2001:10::/28'; do
        ${IPTABLES} -A ${CHAIN} ${TARGET} ${BLOCKED} -j DROP
    done
done

done

# Accept All Nodes link-local scope multicast prefix [RFC4291]
${IPTABLES} -A INPUT -d ff02::1 -j ACCEPT

# Accept All Routers link-local scope
# multicast prefix [RFC4291]
${IPTABLES} -A INPUT -d ff02::2 -j ACCEPT

# Accept Solicited Node link-local scope
# multicast prefix [RFC4291]
${IPTABLES} -A INPUT -d ff02::1:ff00:0000/104 -j ACCEPT

# Accept local scope ICMPv6 packets from
# link-local prefix [RFC4890]
for IP in 'fe80::/64'; do
    # Destination Unreachable [All codes] (Type 1)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        destination-unreachable -d ${IP} -j ACCEPT
    # Packet Too Big (Type 2)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
        packet-too-big -d ${IP} -j ACCEPT
done

```



```
# Time Exceeded [Code 0] (Type 3)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    ttl-zero-during-transit -d ${IP} -j ACCEPT
# Time Exceeded [Code 1] (Type 3)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    ttl-zero-during-reassembly -d ${IP} -j ACCEPT
# Parameter Problem [Code 0] (Type 4)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    bad-header -d ${IP} -j ACCEPT
# Parameter Problem [Code 1] (Type 4)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    unknown-header-type -d ${IP} -j ACCEPT
# Parameter Problem [Code 2] (Type 4)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    unknown-option -d ${IP} -j ACCEPT

# Echo Request (Type 128)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    echo-request -d ${IP} -j ACCEPT
# Echo Response (Type 129)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    echo-reply -d ${IP} -j ACCEPT

# Router Solicitation (Type 133)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 133 \
    -d ${IP} -j ACCEPT
# Router Advertisement (Type 134)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 134 \
    -d ${IP} -j ACCEPT
# Neighbor Solicitation (Type 135)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 135 \
    -d ${IP} -j ACCEPT
```

```
# Neighbor Advertisement (Type 136)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 136 \
    -d ${IP} -j ACCEPT

# Inverse Neighbor Discovery Solicitation (Type 141)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 141 \
    -d ${IP} -j ACCEPT

# Inverse Neighbor Discovery Advertisement (Type 142)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 142 \
    -d ${IP} -j ACCEPT

# Listener Query (Type 130)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 130 \
    -d ${IP} -j ACCEPT

# Listener Report (Type 131)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 131 \
    -d ${IP} -j ACCEPT

# Listener Done (Type 132)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 132 \
    -d ${IP} -j ACCEPT

# Listener Report v2 (Type 143)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 143 \
    -d ${IP} -j ACCEPT

# Certification Path Solicitation (Type 148)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 148 \
    -d ${IP} -j ACCEPT

# Certification Path Advertisement (Type 149)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 149 \
    -d ${IP} -j ACCEPT

# Multicast Router Advertisement (Type 151)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 151 \
    -d ${IP} -j ACCEPT

# Multicast Router Solicitation (Type 152)
```

```

    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 152 \
        -d ${IP} -j ACCEPT
    # Multicast Router Termination (Type 153)
    ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 153 \
        -d ${IP} -j ACCEPT
done

for IP in ${GLOBAL}; do
    for ALLOCATED in ${IANA_PREFIXES}; do
        # Accept routable ICMPv6 packets from
        # IANA allocated prefixes [RFC4890]
        # Destination Unreachable [All codes] (Type 1)
        ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
            destination-unreachable -s ${ALLOCATED} \
            -d ${IP} -j ACCEPT
        # Packet Too Big (Type 2)
        ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
            packet-too-big -s ${ALLOCATED} \
            -d ${IP} -j ACCEPT
        # Time Exceeded [Code 0] (Type 3)
        ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
            ttl-zero-during-transit -s ${ALLOCATED} \
            -d ${IP} -j ACCEPT
        # Time Exceeded [Code 1] (Type 3)
        ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
            ttl-zero-during-reassembly -s ${ALLOCATED} \
            -d ${IP} -j ACCEPT
        # Parameter Problem [Code 0] (Type 4)
        ${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
            bad-header -s ${ALLOCATED} -d ${IP} -j ACCEPT
    done
done

```

```
# Parameter Problem [Code 1] (Type 4)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    unknown-header-type -s ${ALLOCATED} \
    -d ${IP} -j ACCEPT

# Parameter Problem [Code 2] (Type 4)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    unknown-option -s ${ALLOCATED} \
    -d ${IP} -j ACCEPT

# Echo Request (Type 128)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    echo-request -s ${ALLOCATED} -d ${IP} -j ACCEPT

# Echo Response (Type 129)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type \
    echo-reply -s ${ALLOCATED} -d ${IP} -j ACCEPT

# Router Solicitation (Type 133)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 133 \
    -d ${IP} -j ACCEPT

# Router Advertisement (Type 134)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 134 \
    -d ${IP} -j ACCEPT

# Neighbor Solicitation (Type 135)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 135 \
    -d ${IP} -j ACCEPT

# Neighbor Advertisement (Type 136)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 136 \
    -d ${IP} -j ACCEPT

# Inverse Neighbor Discovery Solicitation (Type 141)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 141 \
    -d ${IP} -j ACCEPT
```

```
# Inverse Neighbor Discovery Advertisement (Type 142)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 142 \
    -d ${IP} -j ACCEPT

# Listener Query (Type 130)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 130 \
    -d ${IP} -j ACCEPT

# Listener Report (Type 131)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 131 \
    -d ${IP} -j ACCEPT

# Listener Done (Type 132)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 132 \
    -d ${IP} -j ACCEPT

# Listener Report v2 (Type 143)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 143 \
    -d ${IP} -j ACCEPT

# Certification Path Solicitation (Type 148)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 148 \
    -d ${IP} -j ACCEPT

# Certification Path Advertisement (Type 149)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 149 \
    -d ${IP} -j ACCEPT

# Multicast Router Advertisement (Type 151)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 151 \
    -d ${IP} -j ACCEPT

# Multicast Router Solicitation (Type 152)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 152 \
    -d ${IP} -j ACCEPT

# Multicast Router Termination (Type 153)
${IPTABLES} -A INPUT -p icmpv6 --icmpv6-type 153 \
    -d ${IP} -j ACCEPT
```

```

# traceroute
${IPTABLES} -A INPUT -p udp -m udp -s ${ALLOCATED} \
    -d ${IP} --dport 33434:33523 -m state \
    --state NEW -j REJECT --reject-with \
    icmp6-port-unreachable

done
done

# Accept incoming packets from IANA allocated prefixes and
# connection state is ESTABLISHED or RELATED
for ALLOCATED in ${IANA_PREFIXES}; do
    ${IPTABLES} -A INPUT -s ${ALLOCATED} -m state \
        --state RELATED,ESTABLISHED -j ACCEPT
done

# Send packets from ::/128, link-local and
# global unicast addresses
for IP in '::/128' 'fe80::/64' ${GLOBAL}; do
    ${IPTABLES} -A OUTPUT -s ${IP} -j ACCEPT
done

# Accepted ICMPv6 packets for forwarding [RFC4890]
for SOURCE in ${LAN} ${IANA_PREFIXES}; do
    # Echo Request (Type 128)
    ${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type echo-request \
        -s ${SOURCE} -j ACCEPT
    # Echo Response (Type 129)
    ${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type echo-reply \
        -s ${SOURCE} -j ACCEPT

    # Destination Unreachable [All codes] (Type 1)
    ${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type \
        destination-unreachable -s ${SOURCE} -j ACCEPT
    # Packet Too Big (Type 2)
## Modify the line below -- 1 of 1 ##

```

```

    ${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type \
        packet-too-big -s ${SOURCE} -j DROP
    # Time Exceeded [Code 0] (Type 3)
    ${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type \
        ttl-zero-during-transit -s ${SOURCE} -j ACCEPT
    # Time Exceeded [Code 1] (Type 3)
    ${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type \
        ttl-zero-during-reassembly -s ${SOURCE} -j ACCEPT
    # Parameter Problem [Code 0] (Type 4)
    ${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type \
        bad-header -s ${SOURCE} -j ACCEPT
    # Parameter Problem [Code 1] (Type 4)
    ${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type \
        unknown-header-type -s ${SOURCE} -j ACCEPT
    # Parameter Problem [Code 2] (Type 4)
    ${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type \
        unknown-option -s ${SOURCE} -j ACCEPT
done

# Forward packets which source address belongs to
# internal prefix incoming through eth0 interface
for INTERNAL in ${LAN}; do
    ${IPTABLES} -A FORWARD -s ${INTERNAL} -i eth0 -j ACCEPT
done

# Forward packets which source address belongs to
# IANA allocated prefixes incoming through eth1 interface and
# connection state is ESTABLISHED or RELATED
for ALLOCATED in ${IANA_PREFIXES}; do
    ${IPTABLES} -A FORWARD -s ${ALLOCATED} -m state \
        --state RELATED,ESTABLISHED -i eth1 -j ACCEPT
done

# Allow external nodes to traceroute into LAN
for INTERNAL in ${LAN}; do
    # traceroute

```

```
        ${IPTABLES} -A FORWARD -p udp -m udp -d ${INTERNAL} \  
            --dport 33434:33523 -j ACCEPT  
done  
  
}  
  
stop () {  
    echo "Cleaning 'basename ${IPTABLES}' rules."  
    ${IPTABLES} -P INPUT ACCEPT  
    ${IPTABLES} -F INPUT  
    ${IPTABLES} -P OUTPUT ACCEPT  
    ${IPTABLES} -F OUTPUT  
    ${IPTABLES} -P FORWARD ACCEPT  
    ${IPTABLES} -F FORWARD  
}  
  
status () {  
    ${IPTABLES} --list -v  
}  
  
case "${1}" in  
    start)  
        start  
        ;;  
    stop)  
        stop  
        ;;  
    *)  
        ;;  
esac
```



```

try|test)
    start
    sleep 10
    stop
    ;;
restart|reload|force-reload)
    stop
    sleep 2
    start
    ;;
status)
    status
    ;;
*)
    echo "Usage: ${0} {start|stop|restart|status|try}" >&2
    exit 1
    ;;
esac

exit 0

```

Este *script* também foi escrito com base na RFC 4890 (Davies e Mohacsi, 2007). Quando comparado ao mostrado no passo 3, destacam-se a escuta do endereço *multicast all-routers* e as regras referentes ao encaminhamento (*FORWARD*) dos pacotes permitidos na rede interna de origem local e global, por meio dos prefixos de rede interna, representados pela constante LAN (2001:db8:1dea::/48 e 2001:db8::2/127).

- (c) Ainda no terminal de n1Router, inicialize o serviço de *firewall*:

```
# ./firewall.sh start
```

O resultado do comando é representado pela Figura 3.14.

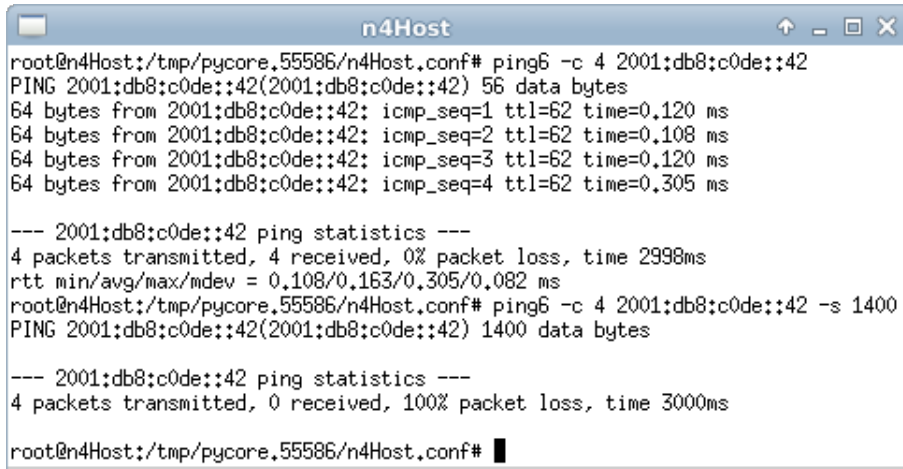


Figura 3.14: *inicialização do serviço de firewall em n1Router.*

- (d) Abra um terminal de n4Host com um duplo-clique e verifique o efeito da regra para derrubar mensagem ICMPv6 tipo 2 (*packet too big*):

```
# ping6 -c 4 2001:db8:c0de::42
# ping6 -c 4 2001:db8:c0de::42 -s 1400
```

O resultado do comando é representado pela Figura 3.15.



```
root@n4Host:/tmp/pycore.55586/n4Host.conf# ping6 -c 4 2001:db8:c0de::42
PING 2001:db8:c0de::42(2001:db8:c0de::42) 56 data bytes
64 bytes from 2001:db8:c0de::42: icmp_seq=1 ttl=62 time=0.120 ms
64 bytes from 2001:db8:c0de::42: icmp_seq=2 ttl=62 time=0.108 ms
64 bytes from 2001:db8:c0de::42: icmp_seq=3 ttl=62 time=0.120 ms
64 bytes from 2001:db8:c0de::42: icmp_seq=4 ttl=62 time=0.305 ms

--- 2001:db8:c0de::42 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.108/0.163/0.305/0.082 ms
root@n4Host:/tmp/pycore.55586/n4Host.conf# ping6 -c 4 2001:db8:c0de::42 -s 1400
PING 2001:db8:c0de::42(2001:db8:c0de::42) 1400 data bytes

--- 2001:db8:c0de::42 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3000ms

root@n4Host:/tmp/pycore.55586/n4Host.conf# █
```

Figura 3.15: verificação do bloqueio de mensagem ICMPv6 *packet too big* para a rede interna.

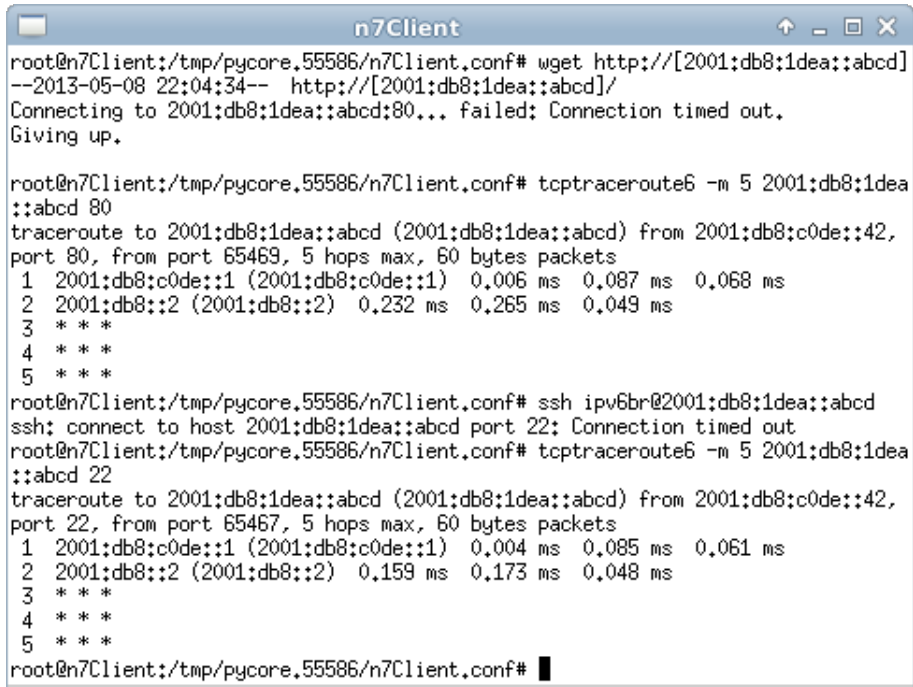
Note que não há nenhum tipo de restrição, uma vez que as políticas de recebimento (*INPUT*), envio (*OUTPUT*) e encaminhamento (*FORWARD*) estão configuradas para aceite (*ACCEPT*), e que há conectividade entre n4Host e n7Client, como se pode verificar no resultado do primeiro ping6. Já o segundo ping6 foi mal sucedido pois no *script* de *firewall* existe uma regra de bloqueio do encaminhamento de pacotes ICMPv6 do tipo *packet too big* destinados à rede interna. O envio de tal mensagem é relacionado ao PMTUD e o funcionamento do protocolo relacionado pode ser verificado na Experiência 1.10.

- (e) Abra um terminal de n7Client com um duplo-clique e verifique o acesso aos serviços HTTP e SSH de n4Host. Para isto utilize os seguintes comandos:

```
# wget http://[2001:db8:1dea::abcd]
# tcptraceroute6 -m 5 2001:db8:1dea::abcd 80
```

```
# ssh ipv6br@2001:db8:1dea::abcd
# tcptraceroute6 -m 5 2001:db8:1dea::abcd 22
```

O resultado dos comandos é representado pela Figura 3.16.



```
root@n7Client:/tmp/pycore.55586/n7Client.conf# wget http://[2001:db8:1dea::abcd]
--2013-05-08 22:04:34-- http://[2001:db8:1dea::abcd]/
Connecting to 2001:db8:1dea::abcd:80... failed: Connection timed out.
Giving up.

root@n7Client:/tmp/pycore.55586/n7Client.conf# tcptraceroute6 -m 5 2001:db8:1dea
::abcd 80
traceroute to 2001:db8:1dea::abcd (2001:db8:1dea::abcd) from 2001:db8:c0de::42,
port 80, from port 65469, 5 hops max, 60 bytes packets
 1 2001:db8:c0de::1 (2001:db8:c0de::1) 0,006 ms 0,087 ms 0,068 ms
 2 2001:db8::2 (2001:db8::2) 0,232 ms 0,265 ms 0,049 ms
 3 * * *
 4 * * *
 5 * * *

root@n7Client:/tmp/pycore.55586/n7Client.conf# ssh ipv6br@2001:db8:1dea::abcd
ssh: connect to host 2001:db8:1dea::abcd port 22: Connection timed out
root@n7Client:/tmp/pycore.55586/n7Client.conf# tcptraceroute6 -m 5 2001:db8:1dea
::abcd 22
traceroute to 2001:db8:1dea::abcd (2001:db8:1dea::abcd) from 2001:db8:c0de::42,
port 22, from port 65467, 5 hops max, 60 bytes packets
 1 2001:db8:c0de::1 (2001:db8:c0de::1) 0,004 ms 0,085 ms 0,061 ms
 2 2001:db8::2 (2001:db8::2) 0,159 ms 0,173 ms 0,048 ms
 3 * * *
 4 * * *
 5 * * *

root@n7Client:/tmp/pycore.55586/n7Client.conf# █
```

Figura 3.16: acessos por SSH e HTTP a n4Host negados.

Verificando as regras atribuídas no `iptables`, nota-se que o acesso às portas 80 e 22 de `n4Host` (`2001:db8:1dea::abcd`) não está liberado. Nos próximos passos, edite o `firewall` em `n1Router` para resolver essa questão. Modifique também a regra de bloqueio do encaminhamento de pacotes ICMPv6 *packet too big*.

- (f) Abra um terminal de n1Router e edite o arquivo firewall.sh. Insira o trecho em **negrito** e remova o trecho ~~neste formato~~:

```
...
# Packet Too Big (Type 2)
## Modify the line below -- 1 of 1 ##
${IPTABLES} -A FORWARD -p icmpv6 --icmpv6-type \
    packet-too-big -s ${SOURCE} -j DROP ACCEPT
# Time Exceeded [Code 0] (Type 3)
...
# Allow external nodes to traceroute into LAN
for INTERNAL in ${LAN}; do
    # traceroute
    ${IPTABLES} -A FORWARD -p udp -m udp -d ${INTERNAL} \
        --dport 33434:33523 -j ACCEPT
done

# ssh
${IPTABLES} -A FORWARD -p tcp -m tcp \
    -d 2001:db8:1dea::abcd/128 --dport 22 -j ACCEPT
# http
${IPTABLES} -A FORWARD -p tcp -m tcp \
    -d 2001:db8:1dea::abcd/128 --dport 80 -j ACCEPT
}

stop () {
...

```

Note que diversas linhas do *script* foram quebradas através do caractere de barra invertida (\) usado para escape. Ao digitar os comandos em **negrito** no *script*, **não deve constar** nenhum caractere entre a barra invertida e a quebra de linha, inclusive o de espaço em branco.

- (g) Ainda no terminal de n1Router, reinicie o serviço de *firewall*:

```
# ./firewall.sh stop
# ./firewall.sh start
```

O resultado dos comandos é representado pela Figura 3.17.



```
n1Router
root@n1Router:/tmp/pycore.55586/n1Router.conf# ./firewall.sh stop
Cleaning iptables rules.
root@n1Router:/tmp/pycore.55586/n1Router.conf# ./firewall.sh start
root@n1Router:/tmp/pycore.55586/n1Router.conf# █
```

Figura 3.17: reiniciando o serviço de *firewall* em n1Router.

- (h) Abra um terminal de n7Client e verifique o acesso aos serviços HTTP e SSH de n4Host. Execute os comandos:

```
# wget http://[2001:db8:1dea::abcd]
# ssh ipv6br@2001:db8:1dea::abcd
```

Quando solicitada, a senha de acesso é *ipv6br*.

O resultado dos comandos é representado pela Figura 3.18.



```
n7Client
root@n7Client:/tmp/pycore.55586/n7Client.conf# wget http://[2001:db8:1dea::abcd]
--2013-05-08 22:10:23-- http://[2001:db8:1dea::abcd]/
Connecting to 2001:db8:1dea::abcd:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 177 [text/html]
Saving to: `index.html'

100%[=====] 177      --.-K/s   in 0s

2013-05-08 22:10:23 (2.37 MB/s) - `index.html' saved [177/177]

root@n7Client:/tmp/pycore.55586/n7Client.conf# ssh ipv6br@2001:db8:1dea::abcd
ipv6br@2001:db8:1dea::abcd's password:
Welcome to Ubuntu 12.04.2 LTS (GNU/Linux 3.2.0-41-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/

0 packages can be updated.
0 updates are security updates.

ipv6br@n4Host:~$ █
```

Figura 3.18: acesso por HTTP e SSH a n4Host, vindo de n7Client, passando por n1Router.

Após verificar a conectividade por HTTP e SSH, encerre a sessão:

```
# exit
```

5. Os nós `n3Router` e `n5Client`, do prefixo `2001:db8:1dea:cafe::/64`, apesar de não serem utilizados efetivamente neste experimento, estão com os respectivos *scripts* de *firewall* configurados. Note que o nó `n5Client` é configurado por meio de mensagens RA enviadas por `n3Router` e que eles apresentam as seguintes características:

`n3Router`

Nó no meio do caminho que não atua como roteador de borda.

`n5Client`

Cliente que utiliza a autoconfiguração *stateless* de endereços IPv6.

Estude seus *scripts* e compare com os deste experimento.

6. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 3.3. IPsec: modo de transporte

Objetivo

Esse laboratório tem o objetivo de demonstrar a configuração de uma rede para a utilização de IPsec em modo de Transporte. O IPsec será configurado nos *hosts* n1HostABCD e n4Host1234, garantindo que todo tráfego entre eles seja protegido.

Para o presente exercício será usada a topologia de rede descrita no arquivo: **3-03-IPsec-transport.imn**.

Introdução teórica

Destinado principalmente a interligar redes de pesquisa acadêmicas, o projeto original do IPv4 não apresentava nenhuma grande preocupação com questões relacionadas à segurança das informações transmitidas. No entanto, o aumento da importância da Internet para a realização de transações entre empresas e consumidores, por exemplo, fez com que um nível maior de segurança passasse a ser exigido, como identificação de usuários e criptografia de dados, tornando necessário anexar novos mecanismos ao protocolo original, que garantissem tais serviços.

O IPsec foi criado para suprir esta deficiência. Ele é uma suíte de protocolos que atua como extensão do protocolo IP e oferece serviços de segurança para prover autenticidade, integridade e confidencialidade aos pacotes IP. Os serviços são providos na camada de rede e, portanto, também oferecem proteção às camadas superiores. A sua arquitetura foi originalmente especificada na RFC 2401 (Kent e Atkinson, 1998) em 1998 e posteriormente atualizada pela RFC 4301 (Kent e Seo, 2005) em 2005.

Há dois modos de operação no IPsec, o **modo de transporte** e o **modo túnel**. O primeiro é usado para proteger a conexão entre apenas duas máquinas, enquanto que o segundo pode ser configurado entre roteadores de borda em redes diferentes, protegendo assim todo o tráfego entre estas duas redes.

A estrutura do IPsec baseia-se em dois cabeçalhos:

Authentication Header (AH)

Provê autenticação e integridade dos pacotes.

Encapsulated Security Payload (ESP)

Adiciona confidencialidade por meio da criptografia dos dados a serem enviados.

Outro ponto fundamental para o funcionamento do IPsec é o compartilhamento das chaves utilizadas para autenticação e criptografia. Recomenda-se a utilização do protocolo IKE (*Internet Key Exchange*) para garantir um meio seguro para a troca das chaves entre os dispositivos que utilizam IPsec e essa troca pode ocorrer de dois modos: mediante o uso de chaves pré-compartilhadas; e da utilização de certificados X.509.

O protocolo IKE trabalha em duas fases:

1. Por meio de uma série de mensagens trocadas, a autenticidade dos dispositivos é verificada e uma chave ISAKMP SA (*Internet Security Association Key Management Security Association*) é gerada.
2. A partir da chave ISAKMP SA, as chaves para o AH e ESP para esta comunicação são geradas e o IPsec começa a ser utilizado.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **3-03-IPsec-transport.imn** localizado no diretório lab, dentro do Desktop. A topologia da rede, representada pela Figura 3.19, deve aparecer.

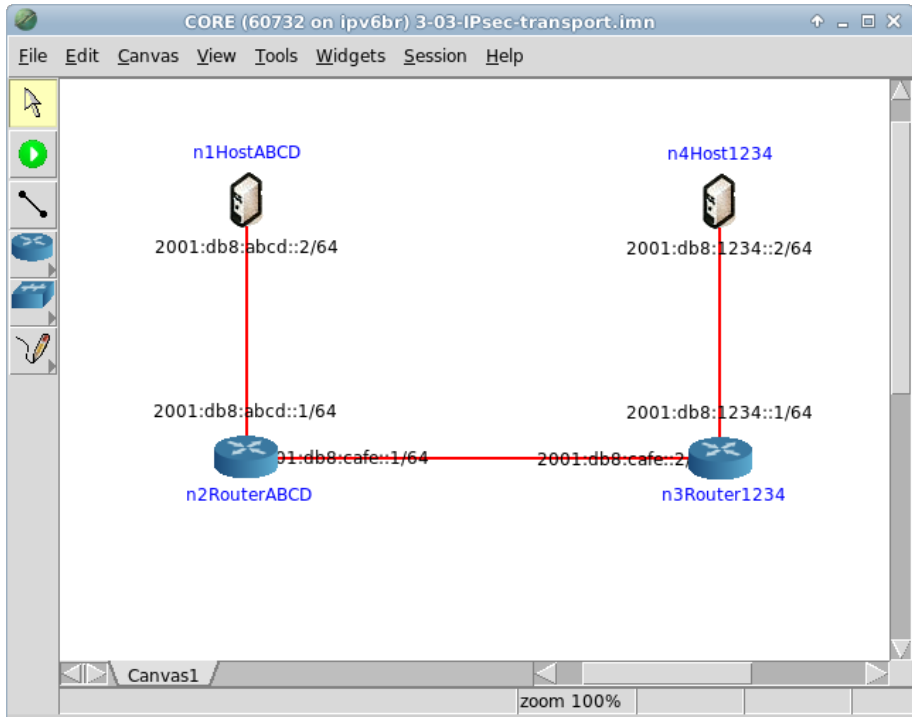


Figura 3.19: topologia da *Experiência 3.3* no CORE.

O objetivo desta topologia de rede é demonstrar que a implantação do IPsec em modo de transporte independe da topologia de rede, isto é, que o IPsec consegue autenticar e criptografar pacotes que são roteados por diversos nós até chegarem ao destino. Neste exemplo, o IPsec será configurado nos *hosts* n1HostABCD e n4Host1234. Note que há dois roteadores intermediando a comunicação entre os *hosts*.

Primeiro serão enviados pacotes de *echo request* sem o IPsec estar configurado. A seguir, será configurada uma conexão com somente autenticação entre os dois *hosts*. Depois disto, irá configurar uma conexão com autenticação e criptografia, capturando pacotes a cada alteração da configuração. Ao final, serão analisados os pacotes capturados nas três situações.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação e verifique a configuração de endereços IPv6 nos nós n1HostABCD e n4Host1234.
3. Faça a captura de pacotes transmitidos entre os *hosts* sem a configuração do IPsec.

4. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface eth1 de n2RouterABCD. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - (b) A verificação de conectividade IPv6 entre n1HostABCD e n4Host1234.

Após efetuar a verificação de conectividade IPv6, encerre a coleta de pacotes trafegados do n2RouterABCD, por meio da combinação de teclas Ctrl+C no terminal em que o tcpdump estiver sendo executado.

5. No terminal do n1HostABCD, gere a chave de autenticação AH executando o comando a seguir. Este comando gerará a chave AH, criará o arquivo chave-ah-h1 e armazenará nele a chave gerada.

```
# dd if=/dev/random count=16 bs=1 | xxd -ps > chave-ah-h1
```

O resultado do comando é representado pela Figura 3.20.



```
n1HostABCD
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# dd if=/dev/random count=16 bs=1 | xxd -ps > chave-ah-h1
16+0 records in
16+0 records out
16 bytes (16 B) copied, 0.000177044 s, 90.4 kB/s
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf#
```

Figura 3.20: geração da chave AH.

6. Gere a chave de criptografia ESP para o mesmo *host*, executando o comando a seguir no terminal. Este comando gerará a chave ESP, criará o arquivo chave-esp-h1 e armazenará nele a chave gerada.

```
# dd if=/dev/random count=24 bs=1 | xxd -ps > chave-esp-h1
```

O resultado do comando é representado pela Figura 3.21.



```
n1HostABCD
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# dd if=/dev/random count=24 bs
=1 | xxd -ps > chave-esp-h1
24+0 records in
24+0 records out
24 bytes (24 B) copied, 4,85155 s, 0,0 kB/s
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# █
```

Figura 3.21: geração da chave ESP.

Após a execução dos comandos anteriores, verifique o conteúdo do arquivo `chave-ah-h1` por meio do comando `cat` e anote o valor apresentado.

```
# cat chave-ah-h1
```

O resultado deve ser similar ao representado na Figura 3.22. Note a extensão da chave AH e os caracteres hexadecimais que a compõem. A chave gerada durante a experiência deve conter a mesma quantidade de caracteres, mas será diferente da chave apresentada na Figura 3.22.



```
n1HostABCD
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# cat chave-ah-h1
d24db54c4617d53b9bac35f4344486a2
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# █
```

Figura 3.22: exemplo de chave AH para o `n1HostABCD`.

7. No terminal do `n4Host1234`, gere a chave de autenticação AH executando o comando a seguir:

```
# dd if=/dev/random count=16 bs=1 | xxd -ps > chave-ah-h2
```

O resultado deve ser similar ao representado na Figura 3.23.

8. Gere a chave de criptografia ESP para o mesmo `host`, executando o comando a seguir no terminal:

```
# dd if=/dev/random count=24 bs=1 | xxd -ps > chave-esp-h2
```

O resultado deve ser similar ao da Figura 3.24.



```
n4Host1234
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# dd if=/dev/random count=16 bs
=1 | xxd -ps > chave-ah-h2
16+0 records in
16+0 records out
16 bytes (16 B) copied, 0.00138761 s, 11.5 kB/s
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# █
```

Figura 3.23: geração de chave AH para o n4Host1234.



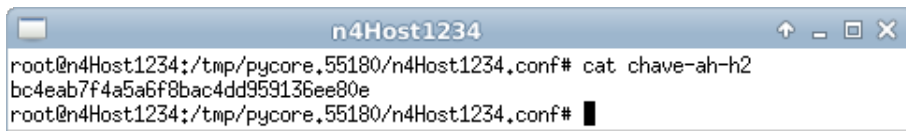
```
n4Host1234
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# dd if=/dev/random count=24 bs
=1 | xxd -ps > chave-esp-h2
24+0 records in
24+0 records out
24 bytes (24 B) copied, 14.5171 s, 0.0 kB/s
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# █
```

Figura 3.24: geração de chave ESP para o n4Host1234.

Após a execução do comando anterior, execute o comando `cat` para verificar o conteúdo do arquivo `chave-ah-h2`. Anote o valor apresentado:

```
# cat chave-ah-h2
```

O resultado deve ser similar ao representado na Figura 3.25.



```
n4Host1234
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# cat chave-ah-h2
bc4eab7f4a5a6f8bac4dd959136ee80e
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# █
```

Figura 3.25: exemplo de chave AH para o n4Host1234.

9. Escreva o arquivo de configuração do IPsec para o n1HostABCD.
 - (a) Abra um terminal de n1HostABCD com um duplo-clique e crie o arquivo de configuração do IPsec:

```
# touch ipsec-h1.conf
```

O resultado do comando é representado pela Figura 3.26.



Figura 3.26: criação do arquivo de configuração do IPsec.

- (b) Ainda no terminal de n1HostABCD, edite o arquivo de configuração do IPsec, localizado em ipsec-h1.conf, de modo a inserir as seguintes linhas:

```
#!/usr/sbin/setkey -f
flush;
spdflush;

add 2001:db8:1234::2 2001:db8:abcd::2 ah 0x300
    -A hmac-md5 0x[chave-ah-h1];
add 2001:db8:abcd::2 2001:db8:1234::2 ah 0x301
    -A hmac-md5 0x[chave-ah-h2];

spdadd 2001:db8:1234::2/64 2001:db8:abcd::2/64 any
    -P in ipsec ah/transport//require;
spdadd 2001:db8:abcd::2/64 2001:db8:1234::2/64 any
    -P out ipsec ah/transport//require;
```

Lembre-se de trocar a parte [chave-ah-h1] e a parte [chave-ah-h2], sem os colchetes, pelas chaves reais, que estão armazenadas nos arquivos chave-ah-h1 e chave-ah-h2.

A primeira linha define o setkey como o interpretador das linhas seguintes no arquivo. Já o comando flush limpa todas as entradas existentes no SAD (*Security Association Database*), enquanto o spdflush limpa todas as entradas do SPD (*Security Policy Database*).

- (c) Logo em seguida há o comando de criação de SA (*Security Association*), uma linha para cada uma das duas SAs (identificado pelo comando add) de autenticação do n1HostABCD. Lembre-se que as SAs são unidirecionais, isto é, cada SA é definida para apenas um sentido da comunicação entre dois pontos de rede, o que torna necessário criar duas SAs para cada comunicação bidirecionalmente segura. As SAs definem também qual

o protocolo a ser usado (nestes exemplos, ou AH ou ESP), um índice único maior que 255 (o índice também pode ser escrito em hexadecimal, usando o prefixo `0x`), o algoritmo usado na criação da chave (os grupos de algoritmos possíveis são diferentes para autenticação e para criptografia) e a chave AH da máquina com o IP de origem. A estrutura do comando que cria uma SA é apresentada a seguir:

```
add [ip_origem] [ip_destino] [protocolo] [indice] [algoritmo] [chave];
```

Nesta experiência, haverá apenas a autenticação entre os *hosts* `n1HostABCD` e `n4Host1234`, portanto será necessário definir apenas duas SAs: uma para a autenticação dos pacotes enviados desta máquina para o `n4Host1234` e outra para a autenticação dos pacotes enviados do `n4Host1234` para esta máquina. Os parâmetros usados serão:

```
[ip_origem]    2001:db8:1234::2
[ip_destino]   2001:db8:abcd::2
[protocolo]    ah
[indice]       0x300 (o valor foi escolhido aleatoriamente)
[algoritmo]    -A hmac-md5
[chave]        chave AH presente no arquivo chave-ah-h1, gerado
                anteriormente. Como este valor é hexadecimal,
                é necessário colocar o prefixo 0x antes do valor.
```

A linha de comando para gerar a segunda SA, no sentido oposto de comunicação, será muito similar à linha anterior, porém com as seguintes diferenças: primeiro, os IPs serão trocados de posição (o IP `abcd::2` será a origem e o IP `1234::2` será o destino); segundo, o índice terá que ser diferente (por exemplo, `0x301`) e; terceiro, a chave AH também será diferente (neste exemplo, a chave será o conteúdo do arquivo `chave-ah-h2`). Não se deve esquecer de colocar o prefixo `0x` antes da chave.

- (d) Depois da definição de SAs, são definidas as políticas de segurança desta máquina. Neste exemplo, será utilizado o seguinte formato simplificado para a política:

```
spdadd [ips_origem] [ips_destino] [protocolo_camada_superior] [politica];
```

As políticas também são unidirecionais e precisam ser definidas aos pares se for preciso uma segurança IPsec nos dois sentidos de comunicação. Os dois primeiros parâmetros são iguais aos das SAs, porém aqui eles podem definir tanto um único IP quanto um intervalo de IPs (por exemplo, pode-se definir como IPs de origem um prefixo como 2001:db8::/64). O terceiro parâmetro, [protocolo_camada_superior], define em quais protocolos das camadas superiores do TCP/IP o IPsec será aplicado (neste caso se usará any, o que significa que esta política será aplicada aos pacotes de todos os protocolos cabíveis). O último parâmetro ([politica]) define o que será feito com os pacotes enviados de [ips_origem] para [ips_destino] e que possuem o protocolo definido em [protocolo_camada_superior]. O campo começa com o parâmetro -P, depois define a direção da comunicação (as opções são out, in e fwd, que significa forward); a seguir define-se as opções de tratamento dos pacotes, que são discard, none e ipsec (será usada esta última) e, por fim, define-se a regra pela qual os pacotes serão processados. Neste exemplo, a regra será ah/transport//require.

Assim, os parâmetros usados para a primeira política serão:

```
[ips_origem]          2001:db8:1234::2
[ips_destino]         2001:db8:abcd::2
[protocolo_camada_superior] any
[politica]            -P in ipsec ah/transport//require
```

A linha de comando para gerar a segunda política, no sentido oposto de comunicação, será muito similar à linha anterior, porém com as seguintes diferenças: primeiro, os IPs serão trocados de posição (o IP abcd::2 será a origem e o IP 1234::2 será o destino) e segundo o campo [politica] terá o sentido out no lugar de in.

- (e) Exiba o conteúdo do arquivo:

```
# cat ipsec-h1.conf
```

O resultado do comando é representado pela Figura 3.27, exceto pelos valores das chaves.



```

root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# cat ipsec-h1.conf
#!/usr/sbin/setkey -f
flush;
spdflush;

add 2001:db8:1234::2 2001:db8:abcd::2 ah 0x300
-A hmac-md5 0xd24db54c4617d53b9bac35f4344486a2;
add 2001:db8:abcd::2 2001:db8:1234::2 ah 0x301
-A hmac-md5 0xbc4eab7f4a5a6f8bac4dd959136ee80e;

spdadd 2001:db8:1234::2/64 2001:db8:abcd::2/64 any
-P in ipsec ah/transport//require;
spdadd 2001:db8:abcd::2/64 2001:db8:1234::2/64 any
-P out ipsec ah/transport//require;
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf#

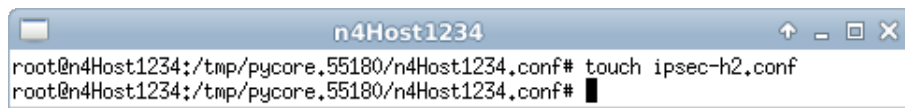
```

Figura 3.27: configuração parcial do IPsec para o n1HostABCD.

10. Escreva o arquivo de configuração do IPsec para o n4Host1234.
 - (a) Abra um terminal de n4Host1234 com um duplo-clique e crie o arquivo de configuração do IPsec. Execute o seguinte comando:

```
# touch ipsec-h2.conf
```

O resultado do comando é representado pela Figura 3.28.



```

root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# touch ipsec-h2.conf
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf#

```

Figura 3.28: criação do arquivo de configuração do IPsec.

- (b) Ainda no terminal de n4Host1234, edite o arquivo de configuração do IPsec, localizado em ipsec-h2.conf, de modo a inserir as seguintes linhas:

```

#!/usr/sbin/setkey -f
flush;
spdflush;

add 2001:db8:1234::2 2001:db8:abcd::2 ah 0x300
-A hmac-md5 0x[chave-ah-h1];
add 2001:db8:abcd::2 2001:db8:1234::2 ah 0x301
-A hmac-md5 0x[chave-ah-h2];

```



```
spdadd 2001:db8:1234::2/64 2001:db8:abcd::2/64 any -P out ipsec
ah/transport//require;
spdadd 2001:db8:abcd::2/64 2001:db8:1234::2/64 any -P in ipsec
ah/transport//require;
```

Lembre-se de trocar a parte [chave-ah-h1] e a parte [chave-ah-h2], sem os colchetes, pelas chaves reais, que estão armazenadas nos arquivos chave-ah-h1 e chave-ah-h2.

Observando os parâmetros anteriores, perceba como as SAs do n4Host1234 serão exatamente iguais às SAs do n1HostABCD. De fato, a SA de saída de pacotes do n1HostABCD é exatamente igual à SA de entrada de pacotes do n4Host1234 e a de entrada de pacotes do n1HostABCD é igual à de saída de pacotes do n4Host1234. Porém, como a ordem de declaração das SAs não importa e as SAs não deixam explícito se o sentido é de entrada ou de saída, basta copiar as SAs do n1HostABCD para o arquivo de configuração de sufixo .conf de n4Host1234.

- (c) Observe também que, ao contrário das SAs, as políticas de segurança SPD dizem explicitamente se o sentido de comunicação é de saída (out) ou de entrada (in). Tal declaração está dentro do parâmetro [politica]. Observe também que, com exceção deste detalhe, as duas políticas do n4Host1234 serão iguais às duas políticas do n1HostABCD. A diferença estará exatamente neste sentido de comunicação, que deve ser trocado nas duas políticas do n4Host1234. Portanto, para definir as políticas do n4Host1234 neste exemplo, basta copiar as políticas do n1HostABCD e inverter o sentido de comunicação (mudar para out onde for in e mudar para in onde for out).
- (d) Exiba o conteúdo do arquivo:

```
# cat ipsec-h2.conf
```

O resultado deve ser similar ao representado na Figura 3.29.



```

root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# cat ipsec-h2.conf
#!/usr/sbin/setkey -f
flush;
spdf flush;

add 2001:db8:1234::2 2001:db8:abcd::2 ah 0x300
-A hmac-md5 0xd24db54c4617d53b9bac35f4344486a2;
add 2001:db8:abcd::2 2001:db8:1234::2 ah 0x301
-A hmac-md5 0xbc4eab7f4a5a6f8bac4dd959136ee80e;

spdadd 2001:db8:1234::2/64 2001:db8:abcd::2/64 any
-P out ipsec ah/transport//require;
spdadd 2001:db8:abcd::2/64 2001:db8:1234::2/64 any
-P in ipsec ah/transport//require;
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf#

```

Figura 3.29: configuração parcial do IPsec para o n4Host1234.

Atente-se para as diferenças entre este arquivo e o arquivo `ipsec-h1.conf`. As únicas diferenças são o sentido de comunicação `in/out` explicitado nas políticas de comunicação. Isto significa que o processo de escrita dos dois arquivos de sufixo `.conf` se limita a compor apenas um deles, copiá-lo para a outra máquina e alterar o `in/out` nas políticas `spdadd`.

11. Abra o terminal do `n1HostABCD` e carregue as configurações do arquivo `ipsec-h1.conf` com o comando a seguir:

```
# setkey -f ipsec-h1.conf
```

Se o arquivo for carregado corretamente, nenhuma mensagem será impressa após a execução do comando `setkey -f`, como mostra a Figura 3.30.



```

root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# setkey -f ipsec-h1.conf
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf#

```

Figura 3.30: carregamento das configurações do IPsec.

Caso surja alguma mensagem, o arquivo possivelmente terá algum erro de sintaxe. Neste caso, verifique novamente os comandos descritos nos passos anteriores.

Para verificar se as chaves foram carregadas, execute o seguinte comando:

```
# setkey -D
```

Este comando exibe as SAs que a máquina possui. Caso as SAs tenham sido corretamente carregadas, o resultado do comando deve ser similar ao representado pela Figura 3.31.



```
n1HostABCD
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# setkey -D
2001:db8:abcd::2 2001:db8:1234::2
  ah mode=transport spi=769(0x00000301) reqid=0(0x00000000)
  A: hmac-md5 bc4eab7f 4a5a6f8b ac4dd959 136ee80e
  seq=0x00000000 replay=0 flags=0x00000000 state=mature
  created: Dec 11 14:31:26 2013    current: Dec 11 14:32:02 2013
  diff: 36(s)    hard: 0(s)    soft: 0(s)
  last:
    hard: 0(s)    soft: 0(s)
  current: 0(bytes)    hard: 0(bytes) soft: 0(bytes)
  allocated: 0    hard: 0 soft: 0
  sadb_seq=1 pid=88 refcnt=0
2001:db8:1234::2 2001:db8:abcd::2
  ah mode=transport spi=768(0x00000300) reqid=0(0x00000000)
  A: hmac-md5 d24db54c 4617d53b 9bac35f4 344486a2
  seq=0x00000000 replay=0 flags=0x00000000 state=mature
  created: Dec 11 14:31:26 2013    current: Dec 11 14:32:02 2013
  diff: 36(s)    hard: 0(s)    soft: 0(s)
  last:
    hard: 0(s)    soft: 0(s)
  current: 0(bytes)    hard: 0(bytes) soft: 0(bytes)
  allocated: 0    hard: 0 soft: 0
  sadb_seq=0 pid=88 refcnt=0
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# █
```

Figura 3.31: associações de segurança do IPsec (SAs) parcialmente configuradas para o n1HostABCD.

Já o comando a seguir exibe as políticas de segurança configuradas:

```
# setkey -DP
```

O resultado do comando é representado pela Figura 3.32.



```

n1HostABCD
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# setkey -DP
2001:db8:abcd::2/64[any] 2001:db8:1234::2/64[any] 255
  out prio def ipsec
  ah/transport//require
  created: Dec 11 14:31:26 2013  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=225 seq=1 pid=89
  refcnt=1
2001:db8:1234::2/64[any] 2001:db8:abcd::2/64[any] 255
  fwd prio def ipsec
  ah/transport//require
  created: Dec 11 14:31:26 2013  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=218 seq=2 pid=89
  refcnt=1
2001:db8:1234::2/64[any] 2001:db8:abcd::2/64[any] 255
  in prio def ipsec
  ah/transport//require
  created: Dec 11 14:31:26 2013  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=208 seq=0 pid=89
  refcnt=1
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# █

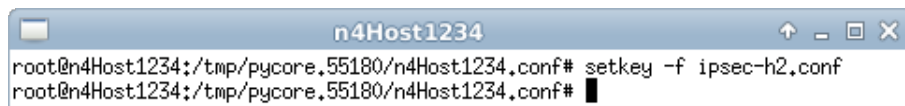
```

Figura 3.32: políticas de segurança do IPsec parcialmente configuradas para o n1HostABCD.

12. Abra o terminal do n4Host1234 e carregue as configurações do arquivo ipsec-h2.conf. Para isto o seguinte comando deve ser utilizado:

```
# setkey -f ipsec-h2.conf
```

Verifique se nenhum erro ocorreu, como mostra a Figura 3.33.



```

n4Host1234
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# setkey -f ipsec-h2.conf
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# █

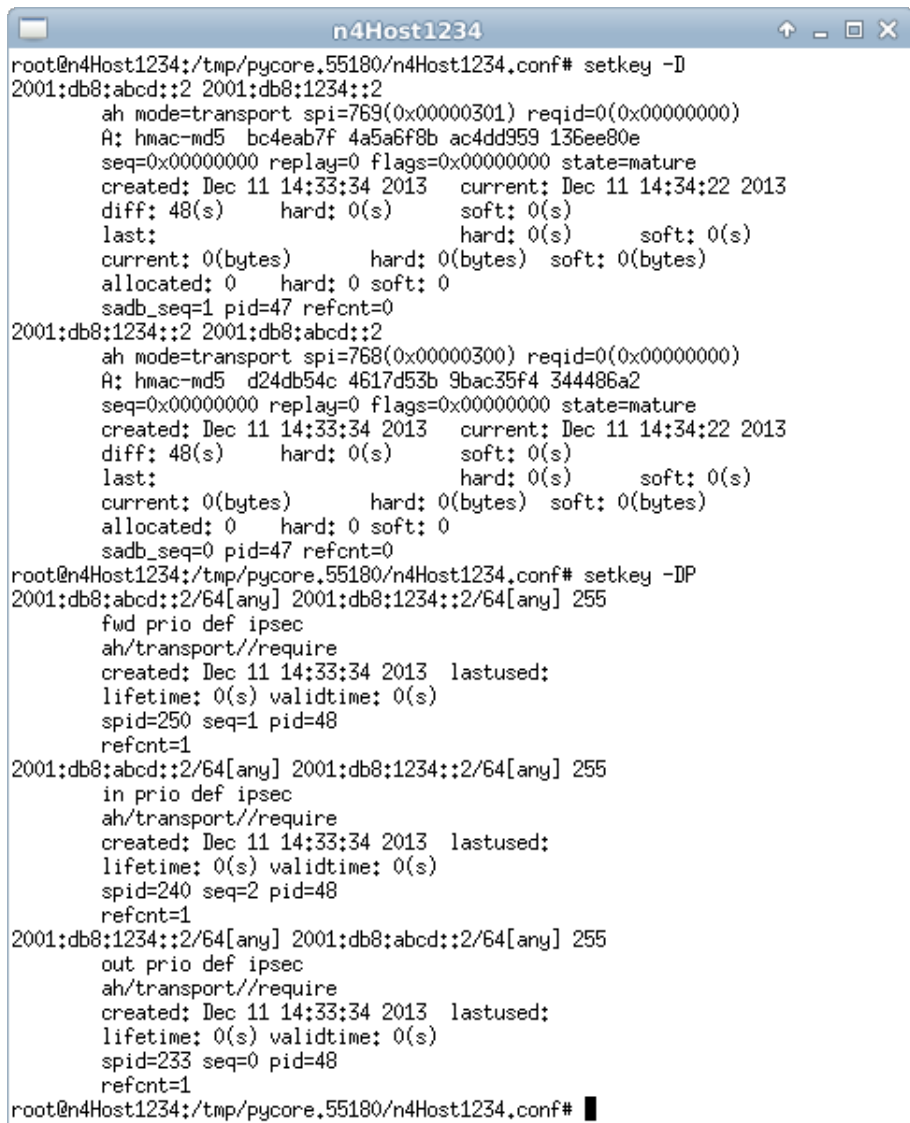
```

Figura 3.33: carregamento das configurações do IPsec.

Para verificar se as chaves foram carregadas, execute os seguintes comandos:

```
# setkey -D
# setkey -DP
```

O resultado deve ser similar ao representado na Figura 3.34.



```
n4Host1234
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# setkey -D
2001:db8:abcd::2 2001:db8:1234::2
  ah mode=transport spi=769(0x00000301) reqid=0(0x00000000)
  A: hmac-md5 bc4eab7f 4a5a6f8b ac4dd959 136ee80e
  seq=0x00000000 replay=0 flags=0x00000000 state=mature
  created: Dec 11 14:33:34 2013   current: Dec 11 14:34:22 2013
  diff: 48(s)   hard: 0(s)   soft: 0(s)
  last:
      hard: 0(s)   soft: 0(s)
  current: 0(bytes)   hard: 0(bytes) soft: 0(bytes)
  allocated: 0   hard: 0 soft: 0
  sadb_seq=1 pid=47 refcnt=0
2001:db8:1234::2 2001:db8:abcd::2
  ah mode=transport spi=768(0x00000300) reqid=0(0x00000000)
  A: hmac-md5 d24db54c 4617d53b 9bac35f4 344486a2
  seq=0x00000000 replay=0 flags=0x00000000 state=mature
  created: Dec 11 14:33:34 2013   current: Dec 11 14:34:22 2013
  diff: 48(s)   hard: 0(s)   soft: 0(s)
  last:
      hard: 0(s)   soft: 0(s)
  current: 0(bytes)   hard: 0(bytes) soft: 0(bytes)
  allocated: 0   hard: 0 soft: 0
  sadb_seq=0 pid=47 refcnt=0
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# setkey -DP
2001:db8:abcd::2/64[any] 2001:db8:1234::2/64[any] 255
  fwd prio def ipsec
  ah/transport//require
  created: Dec 11 14:33:34 2013 lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=250 seq=1 pid=48
  refcnt=1
2001:db8:abcd::2/64[any] 2001:db8:1234::2/64[any] 255
  in prio def ipsec
  ah/transport//require
  created: Dec 11 14:33:34 2013 lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=240 seq=2 pid=48
  refcnt=1
2001:db8:1234::2/64[any] 2001:db8:abcd::2/64[any] 255
  out prio def ipsec
  ah/transport//require
  created: Dec 11 14:33:34 2013 lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=233 seq=0 pid=48
  refcnt=1
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# █
```

Figura 3.34: políticas e associações de segurança do IPsec parcialmente configuradas para o n4Host1234.

13. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface eth1 de n2RouterABCD. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - (b) A verificação de conectividade IPv6 entre n1HostABCD e n4Host1234.

Após efetuar a verificação de conectividade IPv6, encerre a coleta de pacotes trafegados do n2RouterABCD, por meio da combinação de teclas Ctrl+C no terminal em que o tcpdump estiver sendo executado.

Agora será configurada a criptografia entre os dois terminais, em adição à autenticação feita anteriormente. Para isto, altere os arquivos de sufixo .conf já gerados e os recarregue. Em seguida, faça uma terceira captura de pacotes, para análise posterior.

14. Exiba e anote as chaves ESP geradas anteriormente para os *hosts* n1HostABCD e n4Host1234:
 - (a) No terminal do n1HostABCD, exiba a chave ESP armazenada no arquivo chave-esp-h1:

```
# cat chave-esp-h1
```

O resultado deve ser similar ao representado na Figura 3.35.



```

n1HostABCD
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# cat chave-esp-h1
50a9153b7a9467eba3f12d14d8459e0846779429a0795c32
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf#
  
```

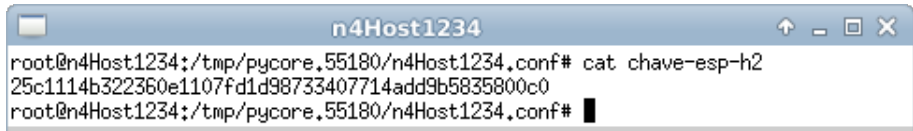
Figura 3.35: exemplo de chave ESP para o n1HostABCD.

Anote o valor apresentado. Perceba como o valor é consideravelmente maior que o valor da chave AH. Contudo, o valor também é hexadecimal, e deverá ter o prefixo 0x dentro do arquivo de sufixo .conf.

- (b) No terminal do n4Host1234, exiba a chave ESP armazenada no arquivo `chave-esp-h2` e anote o valor apresentado.

```
# cat chave-esp-h2
```

O resultado deve ser similar ao representado na Figura 3.36, exceto pelo valor da chave.



```
n4Host1234
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# cat chave-esp-h2
25c1114b322360e1107fd1d98733407714add9b5835800c0
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# █
```

Figura 3.36: *exemplo de chave ESP para o n4Host1234.*

15. Altere o arquivo de configuração `ipsec-h1.conf` para incluir a configuração da criptografia:
- (a) No terminal do n1HostABCD, edite o arquivo de configuração `ipsec-h1.conf`. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. `nano`.

Crie mais um par de SAs correspondentes ao processo de criptografia para o n1HostABCD, inserindo, após o segundo comando `add` já presente no arquivo, mais duas linhas de comando `add`:

```
add 2001:db8:1234::2 2001:db8:abcd::2 esp 0x302 -E 3des-cbc 0x[chave-esp-h1];
add 2001:db8:abcd::2 2001:db8:1234::2 esp 0x303 -E 3des-cbc 0x[chave-esp-h2];
```

Explicando os parâmetros usados:

[ip_origem]	2001:db8:1234::2
[ip_destino]	2001:db8:abcd::2
[protocolo]	esp
[indice]	0x302 (deve ser diferente dos valores já existentes)
[algoritmo]	-E 3des-cbc
[chave]	chave ESP presente no arquivo <code>chave-esp-h1</code> , gerado anteriormente. Este valor também é hexadecimal, logo é necessário colocar o prefixo <code>0x</code> antes da chave.

A linha de comando para gerar a segunda SA, no sentido oposto de comunicação, será muito similar à linha anterior, porém com as seguintes diferenças: primeiro, os IPs serão trocados de posição (o IP `abcd::2` será a origem e o IP `1234::2` será o destino), segundo, o índice terá que ser diferente (por exemplo, `0x303`) e, terceiro, a chave ESP também será diferente (neste exemplo, a chave será o conteúdo do arquivo `chave-esp-h2`).

O arquivo `ipsec-h1.conf` apresentará, neste momento, um conteúdo similar ao representado na Figura 3.37.

```

n1HostABCD
GNU nano 2.2.6 File: ipsec-h1.conf Modified

#!/usr/sbin/setkey -f
flush;
spdflush;

add 2001:db8:1234::2 2001:db8:abcd::2 ah 0x300
-A hmac-md5 0xd24db54c4617d53b9bac35f4344486a2;
add 2001:db8:abcd::2 2001:db8:1234::2 ah 0x301
-A hmac-md5 0xbc4eab7f4a5a6f8bac4dd959136ee80e;

add 2001:db8:1234::2 2001:db8:abcd::2 esp 0x302
-E 3des-cbc 0x50a9153b7a9467eba3f12d14d8459e0846779429a0795c32;
add 2001:db8:abcd::2 2001:db8:1234::2 esp 0x303
-E 3des-cbc 0x25c1114b322360e1107fd1d98733407714add9b5835800c0;

spdadd 2001:db8:1234::2/64 2001:db8:abcd::2/64 any -P in ipsec
ah/transport//require;
spdadd 2001:db8:abcd::2/64 2001:db8:1234::2/64 any -P out ipsec
ah/transport//require;

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

```

Figura 3.37: configuração parcial do IPsec para o `n1HostABCD` (sem as políticas de segurança de criptografia).

- (b) A seguir, atualize as políticas de segurança desta máquina para incluir a criptografia. Ao contrário dos comandos `add`, que definem apenas uma SA cada, os comandos `spdadd` podem definir mais de um protocolo de segurança como parte da política, numa única linha de comando. Para isto edite as linhas que começam com `spdadd` para que fiquem:

```
spdadd 2001:db8:1234::2/64 2001:db8:abcd::2/64 any -P in ipsec
esp/transport//require
ah/transport//require;
spdadd 2001:db8:abcd::2/64 2001:db8:1234::2/64 any -P out ipsec
esp/transport//require
ah/transport//require;
```

Lembrando a estrutura do comando `spdadd`:

```
spdadd [ips_origem] [ips_destino] [protocolo_camada_superior] [politica];
```

A política é definida após o parâmetro `[protocolo_camada_superior]` e apresenta a seguinte forma no arquivo:

```
-P in ipsec
ah/transport//require
```

Para incluir um segundo protocolo de segurança, basta definir o protocolo adicional antes do primeiro, o que fará com que o parâmetro `[politica]` assumira a seguinte forma:

```
-P in ipsec
esp/transport//require
ah/transport//require
```

A ordem em que os protocolos ESP e AH são declarados é importante e deve ser respeitada, isto é, o protocolo ESP deve ser declarado primeiro e em seguida o protocolo AH.

- (c) Exiba o conteúdo do arquivo:

```
# cat ipsec-h1.conf
```

O resultado deve ser similar ao representado na Figura 3.38, exceto pelo valor das chaves.



```

root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# cat ipsec-h1.conf
#!/usr/sbin/setkey -f
flush;
spdflush;

add 2001:db8:1234::2 2001:db8:abcd::2 ah 0x300
-A hmac-md5 0xd24db54c4617d53b9bac35f4344486a2;
add 2001:db8:abcd::2 2001:db8:1234::2 ah 0x301
-A hmac-md5 0xbc4eab7f4a5a6f8bac4dd959136ee80e;

add 2001:db8:1234::2 2001:db8:abcd::2 esp 0x302
-E 3des-cbc 0x50a9153b7a9467eba3f12d14d8459e0846779429a0795c32;
add 2001:db8:abcd::2 2001:db8:1234::2 esp 0x303
-E 3des-cbc 0x25c1114b322360e1107fd1d98733407714add9b5835800c0;

spdadd 2001:db8:1234::2/64 2001:db8:abcd::2/64 any -P in ipsec
esp/transport//require
ah/transport//require;
spdadd 2001:db8:abcd::2/64 2001:db8:1234::2/64 any -P out ipsec
esp/transport//require
ah/transport//require;
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# █

```

Figura 3.38: configuração completa do IPsec para o n1HostABCD.

16. Atualize o arquivo de configuração do IPsec para o n4Host1234 para incluir a criptografia.
 - (a) No terminal do n4Host1234, edite o arquivo de configuração ipsec-h2.conf. No Apêndice C são apresentados alguns editores de texto disponíveis, p. ex. nano.

Após o segundo comando `add` já presente no arquivo, insira mais duas linhas de comando `add`, que definirão o novo par de SAs para a criptografia. Lembrando que o par de SAs é idêntico para os dois *hosts*, copie as SAs de criptografia apresentadas no passo anterior (que serão reapresentadas a seguir):

```

add 2001:db8:1234::2 2001:db8:abcd::2 esp 0x302
-E 3des-cbc 0x[chave-esp-h1];
add 2001:db8:abcd::2 2001:db8:1234::2 esp 0x303
-E 3des-cbc 0x[chave-esp-h2];

```

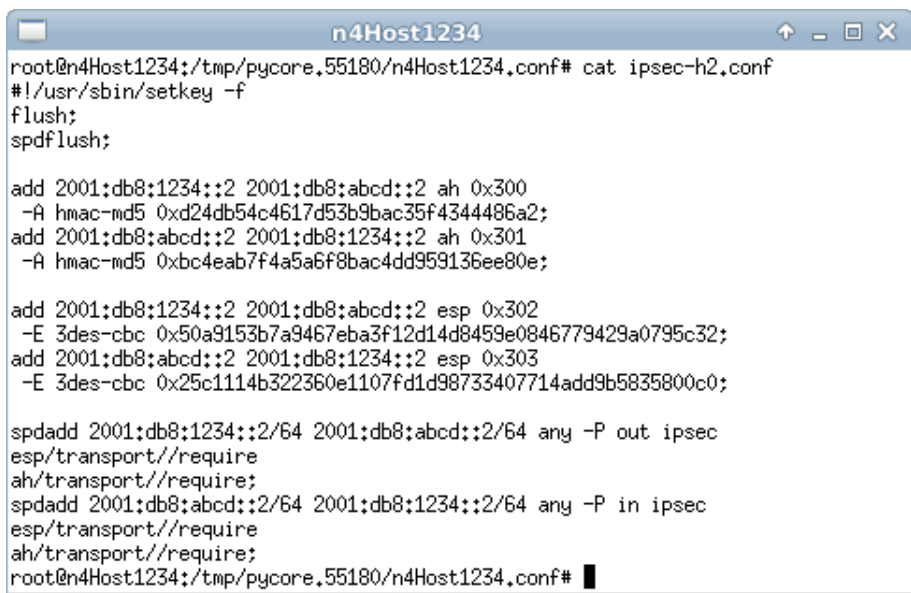
- (b) A seguir, atualize as políticas de segurança desta máquina para incluir a criptografia. Altere os comandos `spdadd` para incluir o texto `esp/transport//require` antes do texto `ah/transport//require`. Da mesma maneira, troque o parâmetro `out` por `in` no primeiro comando e troque o parâmetro `in` por `out` no segundo. Aplique as diferenças marcadas em **negrito** no arquivo de configuração, conforme a seguir:

```
spdadd 2001:db8:1234::2/64 2001:db8:abcd::2/64 any -P out ipsec
esp/transport//require
ah/transport//require;
spdadd 2001:db8:abcd::2/64 2001:db8:1234::2/64 any -P in ipsec
esp/transport//require
ah/transport//require;
```

- (c) Exiba o conteúdo do arquivo:

```
# cat ipsec-h2.conf
```

O resultado deve ser similar ao representado na Figura 3.39, exceto pelo valor das chaves.



```
n4Host1234
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# cat ipsec-h2.conf
#!/usr/sbin/setkey -f
flush;
spdflush;

add 2001:db8:1234::2 2001:db8:abcd::2 ah 0x300
-A hmac-md5 0xd24db54c4617d53b9bac35f4344486a2;
add 2001:db8:abcd::2 2001:db8:1234::2 ah 0x301
-A hmac-md5 0xbc4eab7f4a5a6f8bac4dd959136ee80e;

add 2001:db8:1234::2 2001:db8:abcd::2 esp 0x302
-E 3des-cbc 0x50a9153b7a9467eba3f12d14d8459e0846779429a0795c32;
add 2001:db8:abcd::2 2001:db8:1234::2 esp 0x303
-E 3des-cbc 0x25c1114b322360e1107fd1d98733407714add9b5835800c0;

spdadd 2001:db8:1234::2/64 2001:db8:abcd::2/64 any -P out ipsec
esp/transport//require
ah/transport//require;
spdadd 2001:db8:abcd::2/64 2001:db8:1234::2/64 any -P in ipsec
esp/transport//require
ah/transport//require;
root@n4Host1234:/tmp/pycore.55180/n4Host1234.conf# █
```

Figura 3.39: configuração completa do IPsec para o n4Host1234.

Perceba mais uma vez que a única diferença entre o conteúdo do `ipsec-h2.conf` e o conteúdo do `ipsec-h1.conf` é a inversão da direção de comunicação nas políticas de segurança (onde era `out` torna-se `in` e onde era `in` torna-se `out`).

17. Abra o terminal do `n1HostABCD` e recarregue as configurações do arquivo `ipsec-h1.conf` com o comando a seguir:

```
# setkey -f ipsec-h1.conf
```

Para verificar se as chaves foram carregadas, execute o seguinte comando:

```
# setkey -D
```

O resultado deve ser similar ao representado na Figura 3.40.

Execute também o seguinte comando:

```
# setkey -DP
```

O resultado do comando é representado pela Figura 3.41.

18. Abra o terminal do `n4Host1234` e recarregue as configurações do arquivo `ipsec-h2.conf` com o comando a seguir:

```
# setkey -f ipsec-h2.conf
```

Para verificar se as chaves foram carregadas, execute os seguintes comandos:

```
# setkey -D
```

```
# setkey -DP
```

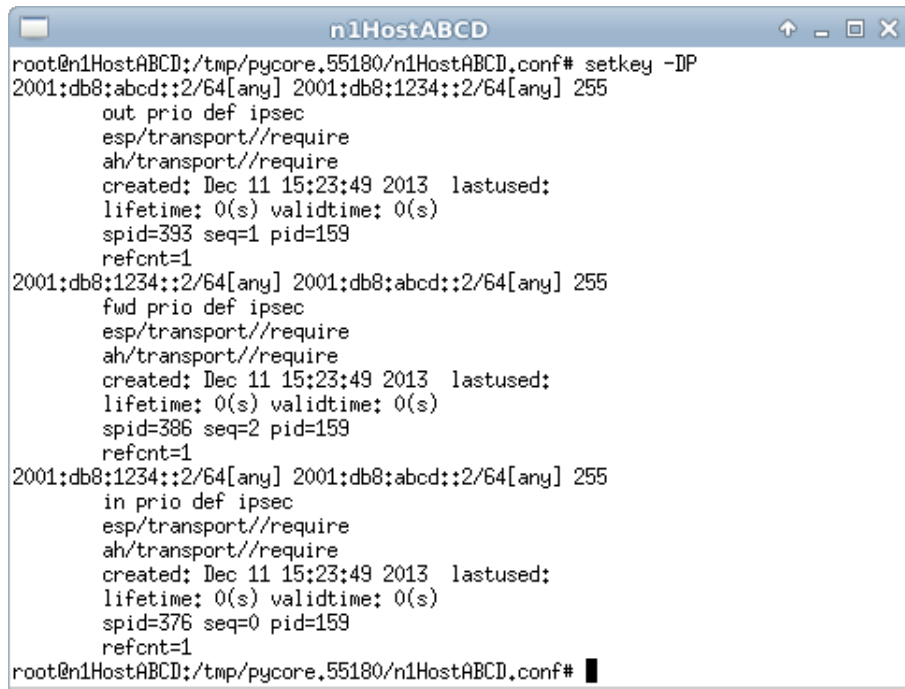
O resultado deve ser similar ao representado na Figura 3.34, visto anteriormente.

```

n1HostABCD
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# setkey -f ipsec-h1.conf
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# setkey -D
2001:db8:abcd::2 2001:db8:1234::2
    esp mode=transport spi=771(0x00000303) reqid=0(0x00000000)
    E: 3des-cbc 25c1114b 322360e1 107fd1d9 87334077 14add9b5 835800c0
    seq=0x00000000 replay=0 flags=0x00000000 state=mature
    created: Dec 11 15:23:49 2013    current: Dec 11 15:23:58 2013
    diff: 9(s)    hard: 0(s)    soft: 0(s)
    last:
    current: 0(bytes)    hard: 0(bytes)    soft: 0(bytes)
    allocated: 0    hard: 0    soft: 0
    sadb_seq=1 pid=157 refcnt=0
2001:db8:1234::2 2001:db8:abcd::2
    esp mode=transport spi=770(0x00000302) reqid=0(0x00000000)
    E: 3des-cbc 50a9153b 7a9467eb a3f12d14 d8459e08 46779429 a0795c32
    seq=0x00000000 replay=0 flags=0x00000000 state=mature
    created: Dec 11 15:23:49 2013    current: Dec 11 15:23:58 2013
    diff: 9(s)    hard: 0(s)    soft: 0(s)
    last:
    current: 0(bytes)    hard: 0(bytes)    soft: 0(bytes)
    allocated: 0    hard: 0    soft: 0
    sadb_seq=2 pid=157 refcnt=0
2001:db8:abcd::2 2001:db8:1234::2
    ah mode=transport spi=769(0x00000301) reqid=0(0x00000000)
    A: hmac-md5 bc4eab7f 4a5a6f8b ac4dd959 136ee80e
    seq=0x00000000 replay=0 flags=0x00000000 state=mature
    created: Dec 11 15:23:49 2013    current: Dec 11 15:23:58 2013
    diff: 9(s)    hard: 0(s)    soft: 0(s)
    last:
    current: 0(bytes)    hard: 0(bytes)    soft: 0(bytes)
    allocated: 0    hard: 0    soft: 0
    sadb_seq=3 pid=157 refcnt=0
2001:db8:1234::2 2001:db8:abcd::2
    ah mode=transport spi=768(0x00000300) reqid=0(0x00000000)
    A: hmac-md5 d24db54c 4617d53b 9bac35f4 344486a2
    seq=0x00000000 replay=0 flags=0x00000000 state=mature
    created: Dec 11 15:23:49 2013    current: Dec 11 15:23:58 2013
    diff: 9(s)    hard: 0(s)    soft: 0(s)
    last:
    current: 0(bytes)    hard: 0(bytes)    soft: 0(bytes)
    allocated: 0    hard: 0    soft: 0
    sadb_seq=0 pid=157 refcnt=0
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# █

```

Figura 3.40: associações de segurança do IPsec (SAs) para o n1HostABCD.



```

n1HostABCD
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# setkey -IP
2001:db8:abcd::2/64[any] 2001:db8:1234::2/64[any] 255
  out prio def ipsec
  esp/transport//require
  ah/transport//require
  created: Dec 11 15:23:49 2013  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=393 seq=1 pid=159
  refcnt=1
2001:db8:1234::2/64[any] 2001:db8:abcd::2/64[any] 255
  fwd prio def ipsec
  esp/transport//require
  ah/transport//require
  created: Dec 11 15:23:49 2013  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=386 seq=2 pid=159
  refcnt=1
2001:db8:1234::2/64[any] 2001:db8:abcd::2/64[any] 255
  in prio def ipsec
  esp/transport//require
  ah/transport//require
  created: Dec 11 15:23:49 2013  lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=376 seq=0 pid=159
  refcnt=1
root@n1HostABCD:/tmp/pycore.55180/n1HostABCD.conf# █

```

Figura 3.41: políticas de segurança do IPsec para o n1HostABCD.

19. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface eth1 de n2RouterABCD. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - (b) A verificação de conectividade IPv6 entre n1HostABCD e n4Host1234.

Após efetuar a verificação de conectividade IPv6, encerre a coleta de pacotes trafegados do n2RouterABCD, por meio da combinação de teclas Ctrl+C no terminal em que o tcpdump estiver sendo executado.

20. Analise agora os pacotes capturados nas três situações:
 - Topologia sem IPsec
 - Topologia somente com autenticação
 - Topologia com autenticação e criptografia

21. Veja se os dados contidos nos pacotes conferem com o que foi passado na teoria.
 - (a) Abra o arquivo contendo a captura da topologia sem IPsec e efetue a análise dos pacotes coletados. Aplique o filtro `icmpv6` no Wireshark e procure pelo pacote *echo reply*, conforme apresentado na Figura 3.42.

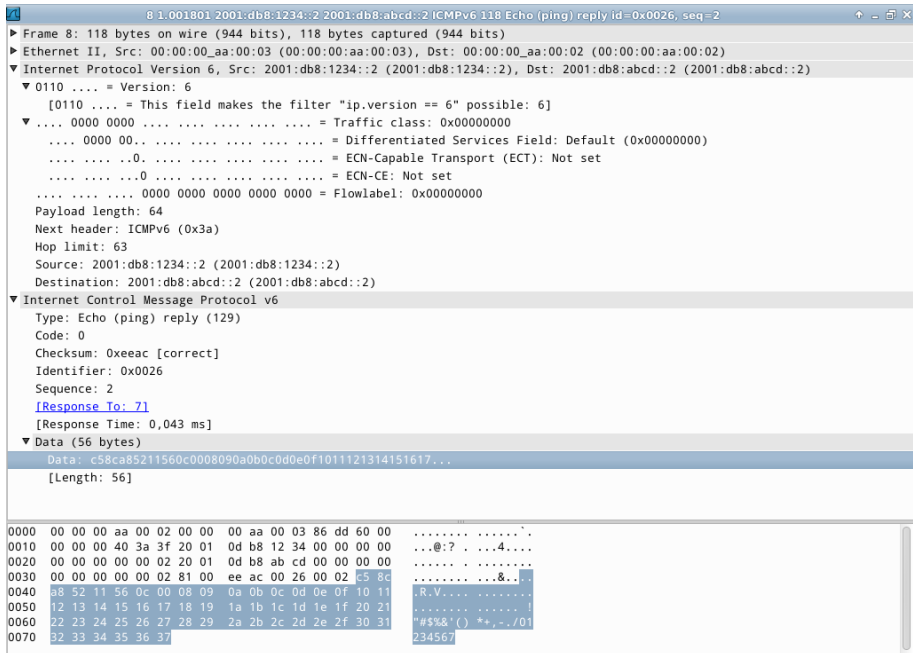


Figura 3.42: captura de pacotes sem IPsec no Wireshark.

- (b) Analise um pacote ICMPv6 de *echo reply* de 2001:db8:1234::2 para 2001:db8:abcd::2. Note que, é possível analisar todo o conteúdo do pacote, inclusive o conteúdo do campo data. Caso o pacote *echo request* fosse falsificado, com o dono se fazendo passar pelo dispositivo 2001:db8:1234::2, a resposta seria enviada mesmo assim.
- (c) Abra o arquivo contendo a captura da topologia somente com autenticação, e efetue a análise dos pacotes coletados. Aplique o filtro `icmpv6` no Wireshark e procure pelo pacote *echo reply*. A estrutura do pacote será similar ao apresentado na Figura 3.43.

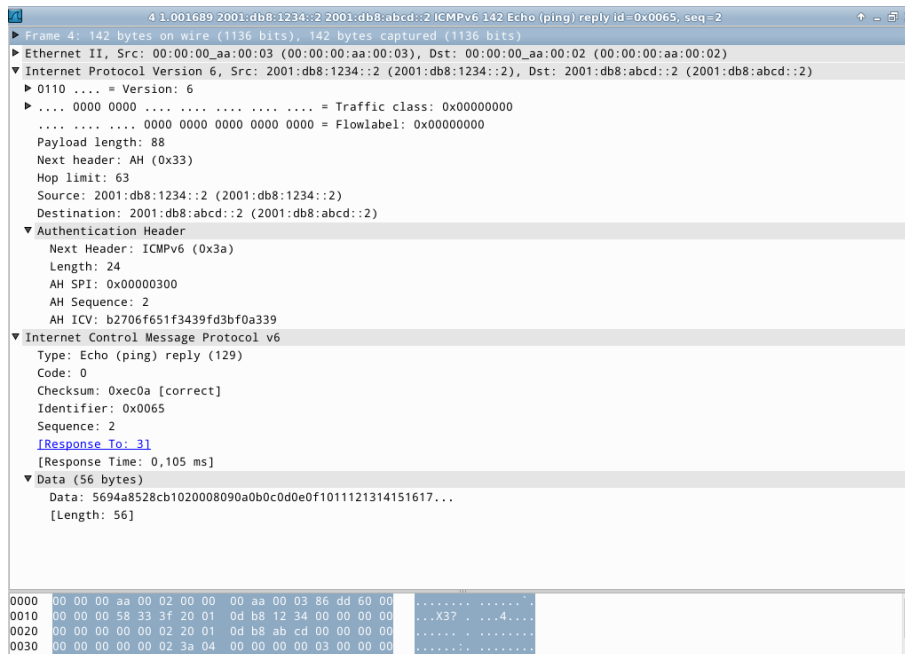


Figura 3.43: captura de pacotes somente com autenticação no Wireshark.

- (d) Analise um pacote de 2001:db8:1234::2 para 2001:db8:abcd::2. Note o cabeçalho de extensão AH que aparece no final do cabeçalho IPv6. Embora o conteúdo do pacote esteja visível para qualquer um que consiga capturá-lo (seja o destinatário do pacote, seja um *sniffer* no meio da rede), o destinatário só responderá o pacote se o remetente possuir a chave de autenticação. Isto significa que, embora os dados não sejam confidenciais, eles são confiáveis e íntegros (garante-se que a origem do pacote não foi forjada e que o pacote não foi modificado). A utilização somente de autenticação se justifica nos casos em que o conteúdo não necessita ser protegido, mas é preciso garantir que o pacote foi enviado por um dispositivo autorizado. Utilizar somente autenticação também é uma técnica utilizada por dispositivos com capacidade de processamento limitada, que não seriam capazes de processar pacotes criptografados em tempo hábil, uma vez que os algoritmos de criptografia necessitam de bastante processamento.
- (e) Abra o arquivo referente a topologia com autenticação e criptografia, conforme representado na Figura 3.44. Aplique o filtro `icmpv6` no Wireshark e procure pelo pacote *echo reply*.

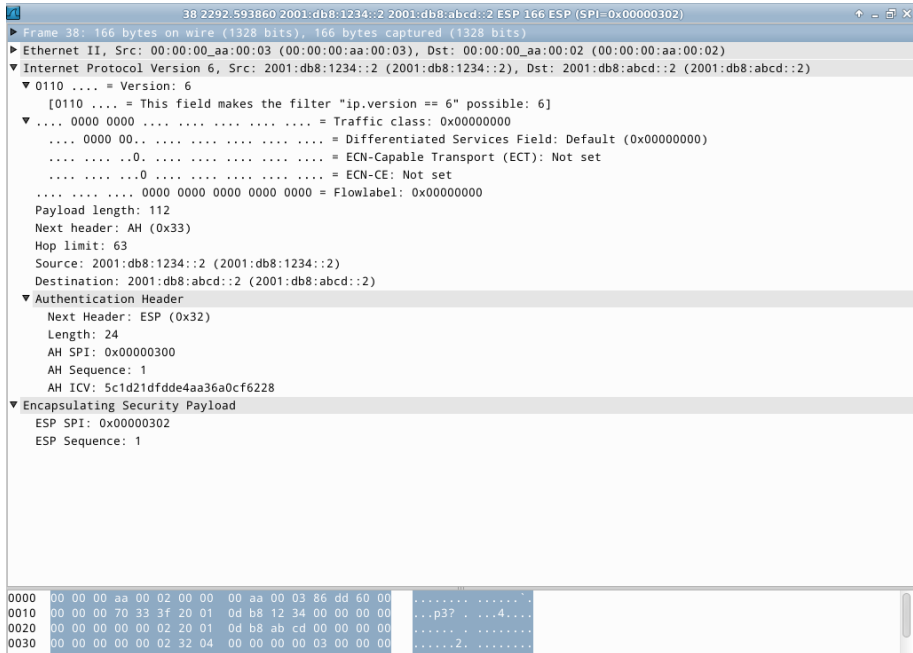


Figura 3.44: captura de pacotes com autenticação e criptografia no Wireshark.

- (f) Note que não é possível analisar todo o conteúdo do pacote. Inclusive, é impossível saber que o pacote em questão é um pacote de *echo reply*. Neste exemplo, sabe-se o tipo do pacote apenas porque força-se a geração dos mesmos por meio de ping6. Outro ponto interessante é que para a camada de aplicação a criptografia dos pacotes é transparente e não afeta seu funcionamento, pois elas recebem e enviam pacotes sem qualquer criptografia ou autenticação, uma vez que estas são geradas e removidas pela camada IP. Note também a existência do cabeçalho de autenticação AH, impedindo que um pacote falsificado seja tratado e respondido pelo destino. Portanto, os dados deste pacote são confidenciais, confiáveis e íntegros.

22. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 3.4. IPsec: modo de túnel

Objetivo

Esse laboratório tem o objetivo de demonstrar a configuração de uma rede para a utilização de IPsec em modo túnel. Será configurado um túnel IPsec entre o `n1RouterABCD` e o `n3Router1234`, fazendo com que todo tráfego entre as redes `2001:db8:abcd::` e `2001:db8:1234::` seja criptografado.

Para o presente exercício será usada a topologia de rede descrita no arquivo: **3-04-IPsec-tunnel.imn**.

Introdução teórica

Ver introdução teórica da Experiência 3.3.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **3-04-IPsec-tunnel.imn** localizado no diretório `lab`, dentro do `Desktop`. A topologia da rede, representada pela Figura 3.45, deve aparecer.

Comparando com a topologia da Experiência 3.3, foram adicionados o `n2Internet` e o dispositivo atacante para simular a Internet, além de um dispositivo cliente para cada roteador da topologia original.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação e verifique a configuração de endereços IPv6 nos nós `n4HostABCD` e `n6Host1234`.
3. Efetue a coleta dos pacotes trafegados na interface `eth0` de `n4HostABCD`. As instruções de coleta de pacotes utilizando `tcpdump` ou Wireshark se encontram no Apêndice C.
4. Deixe o Wireshark aberto, capturando pacotes e abra o terminal da máquina `n5Spoofer` com um duplo-clique. Agora utilize o comando `thcping6` para mandar pacotes para o dispositivo `2001:db8:1234::10`. Mande um pacote com o IP de origem correto (`2001:db8::10`) e um segundo pacote informando como IP de origem o IP do `n4HostABCD` (`2001:db8:abcd::10`):

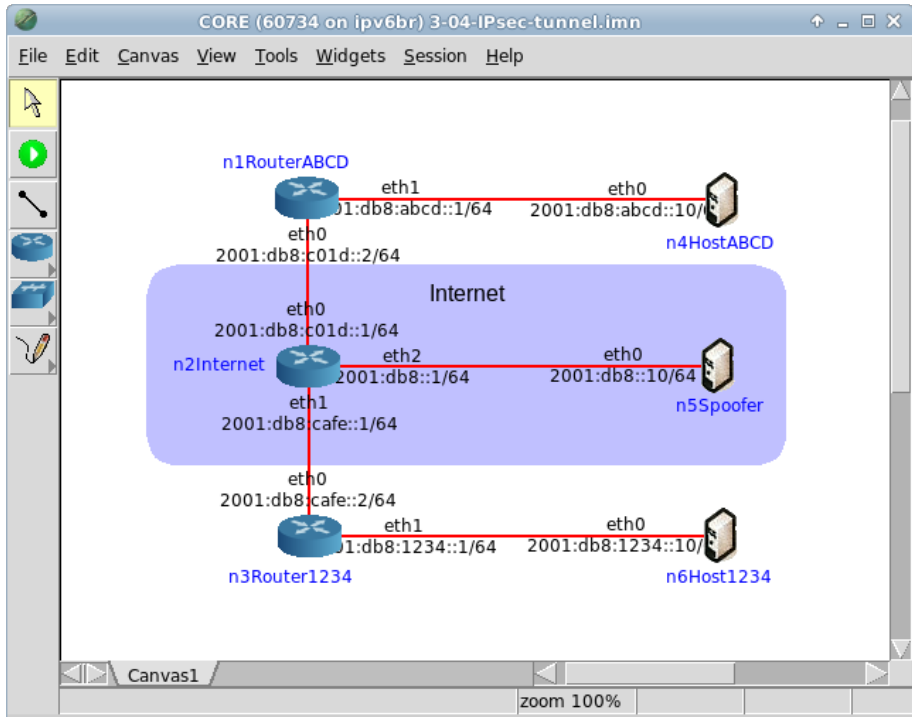


Figura 3.45: topologia da *Experiência 3.4* no CORE.

```
# ping6 -c 4 2001:db8:1234::10
# thcping6 -d 64 eth0 2001:db8::10 2001:db8:1234::10
# thcping6 -d 64 eth0 2001:db8:abcd::10 2001:db8:1234::10
```

O primeiro comando ping6 é necessário para que o programa thcping6 funcione corretamente no CORE.

O resultado deve ser similar ao representado na Figura 3.46.

- Volte ao Wireshark e procure por um pacote do tipo *echo reply* com origem 2001:db8:1234::10 e destino 2001:db8:abcd::10. Pode-se observar que o n4HostABCD recebeu um pacote *echo reply* sem fazer o *echo request*. Esta é a resposta ao *echo request* feito pelo dispositivo atacante com o IP forjado do n4HostABCD. Este tipo de pacote forjado pode ser usado em diversos tipos de ataque, como *man-in-the-middle*, *smurf*, *denial of service* e outros.

```

root@n5Spoofeer:/tmp/pycore.55562/n5Spoofeer.conf# ping6 -c 4 2001:db8:1234::10
PING 2001:db8:1234::10(2001:db8:1234::10) 56 data bytes
64 bytes from 2001:db8:1234::10: icmp_seq=1 ttl=62 time=1.29 ms
64 bytes from 2001:db8:1234::10: icmp_seq=2 ttl=62 time=0.171 ms
64 bytes from 2001:db8:1234::10: icmp_seq=3 ttl=62 time=0.137 ms
64 bytes from 2001:db8:1234::10: icmp_seq=4 ttl=62 time=0.158 ms

--- 2001:db8:1234::10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.137/0.441/1.298/0.494 ms
root@n5Spoofeer:/tmp/pycore.55562/n5Spoofeer.conf# thcping6 -d 64 eth0 2001:db8::1
0 2001:db8:1234::10
0000.0000      ping packet sent to 2001:db8:1234::10
0000.0093      pong packet received from 2001:db8:1234::10
root@n5Spoofeer:/tmp/pycore.55562/n5Spoofeer.conf# thcping6 -d 64 eth0 2001:db8:ab
cd::10 2001:db8:1234::10
0000.0000      ping packet sent to 2001:db8:1234::10
No packet received, terminating.
root@n5Spoofeer:/tmp/pycore.55562/n5Spoofeer.conf# █

```

Figura 3.46: saída do comando thcping6.

6. Para resolver esta falha de segurança, será configurado um túnel IPsec entre o n1RouterABCD e o n3Router1234 fazendo com que todo tráfego entre as redes 2001:db8:abcd:: e 2001:db8:1234:: seja criptografado pela Internet e somente seja aceito na rede destino se possuir cabeçalho de autenticação IPsec válido.

Para configurar o túnel IPsec siga os seguintes passos:

- (a) Abra o terminal de n1RouterABCD com um duplo-clique e verifique o conteúdo do arquivo ipsec.conf com o seguinte comando:

```
# cat ipsec.conf
```

O conteúdo do arquivo é o seguinte:

```
#!/usr/sbin/setkey -f
# Flush the SAD and SPD
flush;
spdflush;

# ESP SAs doing encryption using 192 bit long keys (168 + 24
# parity) and authentication using 128 bit long keys
add 2001:db8:c01d::1 2001:db8:cafe::2 esp 0x201 -m tunnel
    -E 3des-cbc 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831
    -A hmac-md5 0xc0291ff014dccdd03874d9e8e4cdf3e6;

add 2001:db8:cafe::2 2001:db8:c01d::1 esp 0x301 -m tunnel
    -E 3des-cbc 0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df
    -A hmac-md5 0x96358c90783bbfa3d7b196ceabe0536b;

# Security policies
spdadd 2001:db8:abcd::0/64 2001:db8:1234::0/64 any -P out ipsec
    esp/tunnel/2001:db8:c01d::1-2001:db8:cafe::2/require;

spdadd 2001:db8:1234::0/64 2001:db8:abcd::0/64 any -P in ipsec
    esp/tunnel/2001:db8:cafe::2-2001:db8:c01d::1/require;
```

Note que, as chaves deste exemplo já foram criadas e portanto não devem ser utilizadas em aplicações reais. Para gerar suas próprias chaves siga os procedimentos encontrados em Experiência 3.3.

- i. Recarregue as configurações do IPsec para que os dados inseridos no arquivo de configuração sejam executados:

```
# setkey -f ipsec.conf
```

O resultado do comando é representado pela Figura 3.47.



Figura 3.47: carregando as configurações do IPsec.

- ii. Para verificar se as chaves foram carregadas, execute os seguintes comandos:

```
# setkey -D
# setkey -DP
```

A Figura 3.48 representa uma saída similar a deste comando, confirmando os dados recebidos.

- (b) Abra o terminal do n3Router1234, com um duplo-clique. Verifique o conteúdo do arquivo ipsec.conf com o seguinte comando:

```
# cat ipsec.conf
```

O conteúdo do arquivo é o seguinte:

```
#!/usr/sbin/setkey -f
# Flush the SAD and SPD
flush;
spdflush;

# ESP SAs doing encryption using 192 bit long keys (168 + 24
# parity) and authentication using 128 bit long keys

add 2001:db8:c01d::1 2001:db8:cafe::2 esp 0x201 -m tunnel -E
    3des-cbc 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831
    -A hmac-md5 0xc0291ff014dccdd03874d9e8e4cdf3e6;

add 2001:db8:cafe::2 2001:db8:c01d::1 esp 0x301 -m tunnel -E
    3des-cbc 0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df
    -A hmac-md5 0x96358c90783bbfa3d7b196ceabe0536b;

# Security policies

spdadd 2001:db8:abcd::0/64 2001:db8:1234::0/64 any -P in ipsec
    esp/tunnel/2001:db8:c01d::1-2001:db8:cafe::2/require;

spdadd 2001:db8:1234::0/64 2001:db8:abcd::0/64 any -P out ipsec
    esp/tunnel/2001:db8:cafe::2-2001:db8:c01d::1/require;
```

```

n1RouterABCD
root@n1RouterABCD:/tmp/pycore.55562/n1RouterABCD.conf# setkey -D
2001:db8:cafe::2 2001:db8:c01d::2
  esp mode=tunnel spi=769(0x00000301) reqid=0(0x00000000)
  E: 3des-cbc f6ddb555 acfd9d77 b03ea384 3f265325 5afe8eb5 573965df
  A: hmac-md5 96358c90 783bbfa3 d7b196ce abe0536b
  seq=0x00000000 replay=0 flags=0x00000000 state=mature
  created: Dec 12 15:04:48 2013   current: Dec 12 15:05:58 2013
  diff: 70(s)   hard: 0(s)   soft: 0(s)
  last:
  current: 0(bytes)   hard: 0(bytes)   soft: 0(bytes)
  allocated: 0   hard: 0   soft: 0
  sadb_seq=1 pid=45 refcnt=0
2001:db8:c01d::2 2001:db8:cafe::2
  esp mode=tunnel spi=513(0x00000201) reqid=0(0x00000000)
  E: 3des-cbc 7aeaca3f 87d060a1 2f4a4487 d5a5c335 5920fae6 9a96c831
  A: hmac-md5 c0291ff0 14dccdd0 3874d9e8 e4cdf3e6
  seq=0x00000000 replay=0 flags=0x00000000 state=mature
  created: Dec 12 15:04:48 2013   current: Dec 12 15:05:58 2013
  diff: 70(s)   hard: 0(s)   soft: 0(s)
  last:
  current: 0(bytes)   hard: 0(bytes)   soft: 0(bytes)
  allocated: 0   hard: 0   soft: 0
  sadb_seq=0 pid=45 refcnt=0
root@n1RouterABCD:/tmp/pycore.55562/n1RouterABCD.conf# setkey -DP
2001:db8:1234::/64[any] 2001:db8:abcd::/64[any] 255
  fwd prio def ipsec
  esp/tunnel/2001:db8:cafe::2-2001:db8:c01d::2/require
  created: Dec 12 15:04:48 2013   lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=42 seq=1 pid=46
  refcnt=1
2001:db8:1234::/64[any] 2001:db8:abcd::/64[any] 255
  in prio def ipsec
  esp/tunnel/2001:db8:cafe::2-2001:db8:c01d::2/require
  created: Dec 12 15:04:48 2013   lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=32 seq=2 pid=46
  refcnt=1
2001:db8:abcd::/64[any] 2001:db8:1234::/64[any] 255
  out prio def ipsec
  esp/tunnel/2001:db8:c01d::2-2001:db8:cafe::2/require
  created: Dec 12 15:04:48 2013   lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=25 seq=0 pid=46
  refcnt=1
root@n1RouterABCD:/tmp/pycore.55562/n1RouterABCD.conf# █

```

Figura 3.48: *confirmando os dados recebidos.*

- i. Recarregue as configurações do IPsec para que os dados inseridos no arquivo de configuração sejam executados:

```
# setkey -f ipsec.conf
```

O resultado do comando é representado pela Figura 3.49.



Figura 3.49: carregando as configurações do IPsec.

- ii. Para verificar se as chaves foram carregadas, execute os seguintes comandos:

```
# setkey -D
# setkey -DP
```

A Figura 3.50 representa uma saída similar a deste comando, confirmando os dados recebidos.

7. Encerre a captura do Wireshark.
8. Execute a captura de pacotes na interface eth0 de n2Internet. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
9. Abra o console do n4HostABCD e utilize ping6 com destino a n6Host1234:

```
# ping6 -c 4 2001:db8:1234::10
```

O resultado será conforme a Figura 3.51.

10. No Wireshark, procure pacotes do tipo *echo request* ou *echo reply*.
11. Observe que, não é possível encontrar estes tipos de pacotes apesar do comando ping6 ter funcionado corretamente. Isto ocorre porque o túnel estabelecido entre o n1RouterABCD e o n3Router1234 está encriptando os pacotes *echo request* e *echo reply*. Para confirmar isto procure por pacotes com protocolo ESP. É possível notar a existência de quatro pacotes do n1RouterABCD para o n3Router1234 e quatro pacotes do n3Router1234 para o n1RouterABCD. Esta é exatamente a quantidade de pacotes *echo request* e *echo reply* que foi gerada na conversa entre o n4HostABCD e o n6Host1234.

Como o tráfego de rede nesta simulação é controlado, pode-se concluir que estes pacotes realmente são os pacotes *echo request* e *echo reply*. Analise um dos pacotes e note que os IPs de origem e destino são os IPs dos roteadores. A captura do Wireshark será similar à Figura 3.52.


```

n3Router1234
root@n3Router1234:/tmp/pycore.55562/n3Router1234.conf# setkey -D
2001:db8:cafe::2 2001:db8:c01d::2
  esp mode=tunnel spi=769(0x00000301) reqid=0(0x00000000)
  E: 3des-cbc f6ddb555 acfd9d77 b03ea384 3f265325 5afe8eb5 573965df
  A: hmac-md5 96358c90 783bbfa3 d7b196ce abe0536b
  seq=0x00000000 replay=0 flags=0x00000000 state=mature
  created: Dec 12 15:07:02 2013   current: Dec 12 15:07:30 2013
  diff: 28(s)   hard: 0(s)   soft: 0(s)
  last:
  current: 0(bytes)   hard: 0(bytes)   soft: 0(bytes)
  allocated: 0   hard: 0   soft: 0
  sadb_seq=1 pid=45 refcnt=0
2001:db8:c01d::2 2001:db8:cafe::2
  esp mode=tunnel spi=513(0x00000201) reqid=0(0x00000000)
  E: 3des-cbc 7aeaca3f 87d060a1 2f4a4487 d5a5c335 5920fae6 9a96c831
  A: hmac-md5 c0291ff0 14dccdd0 3874d9e8 e4cdf3e6
  seq=0x00000000 replay=0 flags=0x00000000 state=mature
  created: Dec 12 15:07:02 2013   current: Dec 12 15:07:30 2013
  diff: 28(s)   hard: 0(s)   soft: 0(s)
  last:
  current: 0(bytes)   hard: 0(bytes)   soft: 0(bytes)
  allocated: 0   hard: 0   soft: 0
  sadb_seq=0 pid=45 refcnt=0
root@n3Router1234:/tmp/pycore.55562/n3Router1234.conf# setkey -DP
2001:db8:1234::/64[any] 2001:db8:abcd::/64[any] 255
  out prio def ipsec
  esp/tunnel/2001:db8:cafe::2-2001:db8:c01d::2/require
  created: Dec 12 15:07:02 2013   lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=89 seq=1 pid=46
  refcnt=1
2001:db8:abcd::/64[any] 2001:db8:1234::/64[any] 255
  fwd prio def ipsec
  esp/tunnel/2001:db8:c01d::2-2001:db8:cafe::2/require
  created: Dec 12 15:07:02 2013   lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=82 seq=2 pid=46
  refcnt=1
2001:db8:abcd::/64[any] 2001:db8:1234::/64[any] 255
  in prio def ipsec
  esp/tunnel/2001:db8:c01d::2-2001:db8:cafe::2/require
  created: Dec 12 15:07:02 2013   lastused:
  lifetime: 0(s) validtime: 0(s)
  spid=72 seq=0 pid=46
  refcnt=1
root@n3Router1234:/tmp/pycore.55562/n3Router1234.conf# █

```

Figura 3.50: *confirmando os dados recebidos.*

```

n4HostABCD
root@n4HostABCD:/tmp/pycore.48011/n4HostABCD.conf# ping6 -c 4 2001:db8:1234::10
PING 2001:db8:1234::10(2001:db8:1234::10) 56 data bytes
64 bytes from 2001:db8:1234::10: icmp_seq=1 ttl=62 time=0,717 ms
64 bytes from 2001:db8:1234::10: icmp_seq=2 ttl=62 time=0,357 ms
64 bytes from 2001:db8:1234::10: icmp_seq=3 ttl=62 time=0,413 ms
64 bytes from 2001:db8:1234::10: icmp_seq=4 ttl=62 time=0,359 ms

--- 2001:db8:1234::10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0,357/0,461/0,717/0,150 ms
root@n4HostABCD:/tmp/pycore.48011/n4HostABCD.conf#

```

Figura 3.51: teste de conectividade entre clientes de redes diferentes após aplicar o IPsec.

No.	Time	Source	Destination	Protocol	Length	Info
6	16.756976	2001:db8:c01d::2	2001:db8:cafe::2	ESP	194	ESP (SPI=0x00000201)
7	16.757134	2001:db8:cafe::2	2001:db8:c01d::2	ESP	194	ESP (SPI=0x00000301)
8	17.756395	2001:db8:c01d::2	2001:db8:cafe::2	ESP	194	ESP (SPI=0x00000201)
9	17.756568	2001:db8:cafe::2	2001:db8:c01d::2	ESP	194	ESP (SPI=0x00000301)
10	18.756310	2001:db8:c01d::2	2001:db8:cafe::2	ESP	194	ESP (SPI=0x00000201)
11	18.756471	2001:db8:cafe::2	2001:db8:c01d::2	ESP	194	ESP (SPI=0x00000301)
12	20.756218	fe80::200:ff:feaa:1	2001:db8:c01d::2	ICMPv6	86	Neighbor Solicitation for 2001:db8:c01d::2 from
13	20.756399	2001:db8:c01d::2	fe80::200:ff:feaa:1	ICMPv6	78	Neighbor Advertisement 2001:db8:c01d::2 (rtr, sc
14	25.764327	fe80::200:ff:feaa:0	fe80::200:ff:feaa:1	ICMPv6	86	Neighbor Solicitation for fe80::200:ff:feaa:1 fr
15	25.764416	fe80::200:ff:feaa:1	fe80::200:ff:feaa:0	ICMPv6	78	Neighbor Advertisement fe80::200:ff:feaa:1 (rtr,
16	30.772246	fe80::200:ff:feaa:1	fe80::200:ff:feaa:0	ICMPv6	86	Neighbor Solicitation for fe80::200:ff:feaa:0 fr
17	30.772423	fe80::200:ff:feaa:0	fe80::200:ff:feaa:1	ICMPv6	78	Neighbor Advertisement fe80::200:ff:feaa:0 (rtr,

▶ Frame 6: 194 bytes on wire (1552 bits), 194 bytes captured (1552 bits)
 ▶ Ethernet II, Src: 00:00:00_aa:00:00 (00:00:00:aa:00:00), Dst: 00:00:00_aa:00:01 (00:00:00:aa:00:01)
 ▶ Internet Protocol Version 6, Src: 2001:db8:c01d::2 (2001:db8:c01d::2), Dst: 2001:db8:cafe::2 (2001:db8:cafe::2)
 ▶ Encapsulating Security Payload

```

0000 00 00 00 aa 00 01 00 00 00 aa 00 00 86 dd 60 00 .....20.....
0010 00 00 00 8c 32 40 20 01 0d b8 c0 1d 00 00 00 .....Q.....
0020 00 00 00 00 00 02 20 01 0d b8 ca fe 00 00 00 .....
0030 00 00 00 00 00 02 00 00 02 01 00 00 00 02 51 b3 .....

```

Figura 3.52: captura de pacotes trocados entre clientes após configuração do IPsec túnel.

12. Configure o Wireshark para capturar pacotes recebidos no n4HostABCD. Gere um novo pacote *echo request* falsificado no console do atacante, na tentativa de fazer com que o pacote de resposta chegue ao n4HostABCD originado pelo n6Host1234. Para tanto, utilize o comando:

```
# ping6 -c 4 2001:db8:1234::10
# thcping6 -d 64 eth0 2001:db8:abcd::10 2001:db8:1234::10
```

Novamente, o primeiro comando ping6 é necessário para que o programa thcping6 funcione corretamente no CORE.

13. Tente encontrar no Wireshark um pacote do tipo *echo reply* originário do n6Host1234. Veja que não é possível encontrar este pacote, pois ele não chegou ao n4HostABCD.
14. Configure agora o Wireshark para analisar a eth0 do n3Router1234 e repita o envio do pacote forjado. Procure o pacote forjado. É possível ver que ele é recebido pelo n3Router1234, mas um pacote de *echo reply* não passa por esta interface. A captura do Wireshark será similar à Figura 3.53.
15. Configure agora o Wireshark para analisar a eth1 do n3Router1234 e repita o envio do pacote forjado. Veja que o pacote *echo request* chega à interface eth0 do n3Router1234, mas não é redirecionado para a interface eth1 como acontecia quando o IPsec não estava configurado. Isto ocorre porque o roteador recebe um pacote vindo da rede 2001:db8:abcd:: sem estar autenticado e criptografado. Neste caso o comportamento do roteador é descartar o pacote, impedindo o ataque que utiliza a falsificação do endereço de origem.

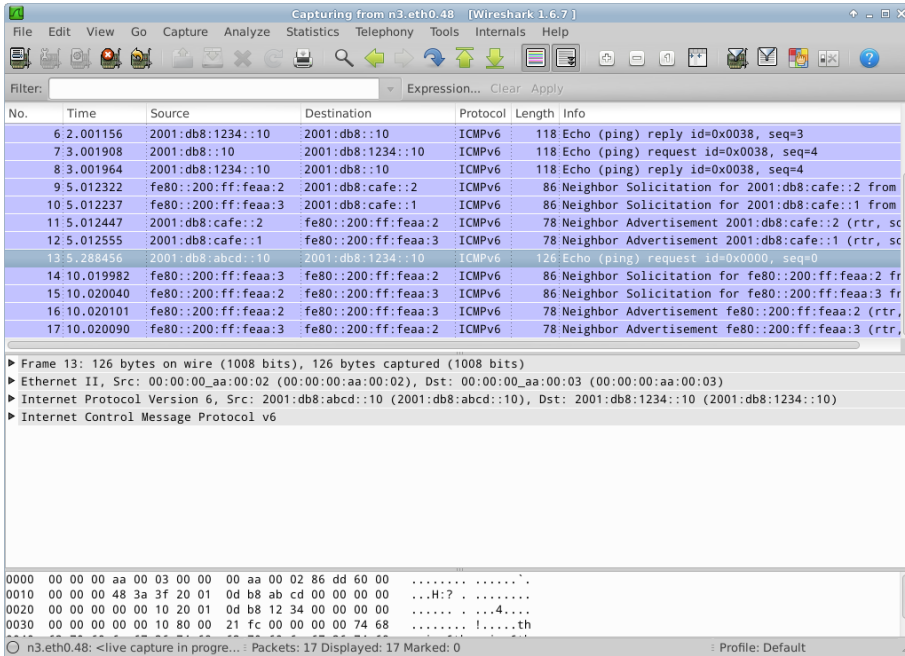


Figura 3.53: captura de pacotes forjados após configuração do IPsec túnel.

16. Encerre a simulação, conforme descrito no Apêndice B.

Capítulo 4

Técnicas de transição

Experiência 4.1. Túnel 6in4

Objetivo

Esta experiência mostra o funcionamento de um túnel manual 6in4, que é capaz de transportar pacotes IPv6 entre dois pontos, através de uma rede IPv4, encapsulando-os. O cenário utilizado tem dois nós pilha dupla e um roteador que funciona somente com IPv4. Um túnel é configurado entre os dois nós e é possível observar como os pacotes IPv6 são transportados por meio do roteador IPv4.

Para o presente exercício será utilizada a topologia descrita no arquivo: **4-01-6in4.imn**.

Introdução teórica

É possível encapsular pacotes IPv6 diretamente dentro de pacotes IPv4, como *payload*. Neste caso, no campo Protocolo do cabeçalho IPv4, especifica-se o valor 41 (29 em hexadecimal). Este tipo de encapsulamento está descrito na RFC 4213 (Nordmark e Gilligan, 2005) e é conhecido como 6in4, ou IPv6-in-IPv4. Popularmente é chamado também de *protocolo 41*.

O encapsulamento é, em si, muito simples. Contudo, ao encapsular um pacote IPv6 dentro de outro IPv4, algumas questões de complexidade maior devem ser tratadas. Por exemplo, pode não haver espaço suficiente para o pacote e deve-se, ou fragmentá-lo, ou devolver uma mensagem ICMPv6 *packet too big* para quem o originou. Deve-se também converter erros ICMPv4 que aconteçam ao longo do caminho em erros ICMPv6.

É possível configurar túneis manualmente, usando o 6in4. Essa configuração consiste basicamente em definir os endereços IPv4 de origem e destino utilizados em cada extremidade do túnel.

Túneis IPv6 estáticos, configurados manualmente, são úteis em diversas situações. Por exemplo, podem ser utilizados para contornar um equipamento ou enlace que não suporta IPv6 numa determinada rede. Podem também interligar duas redes IPv6 por meio da Internet IPv4.

A Figura 4.1 ilustra como o processo de encapsulamento de IPv6 em IPv4 acontece.

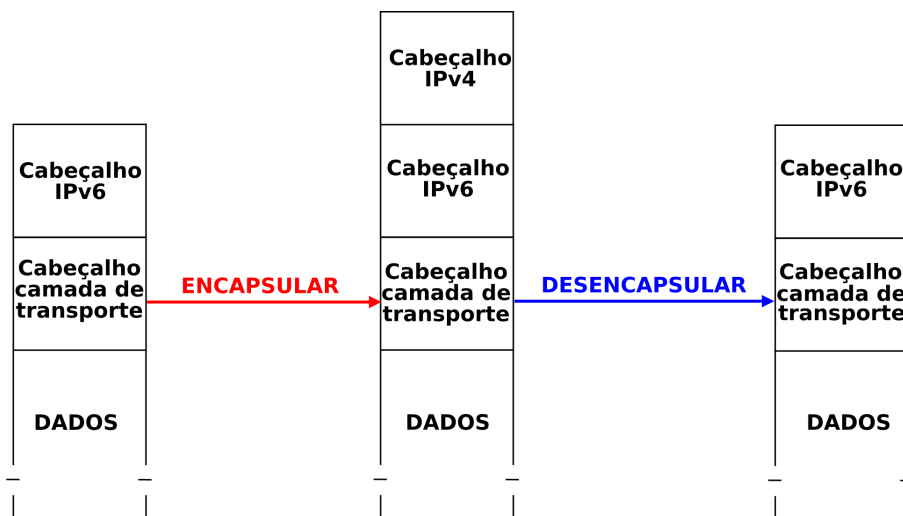


Figura 4.1: Encapsulamento de pacote IPv6 em IPv4 (6in4).

Roteiro experimental

1. Inicie o CORE e abra o arquivo **4-01-6in4.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 4.2, deve aparecer.

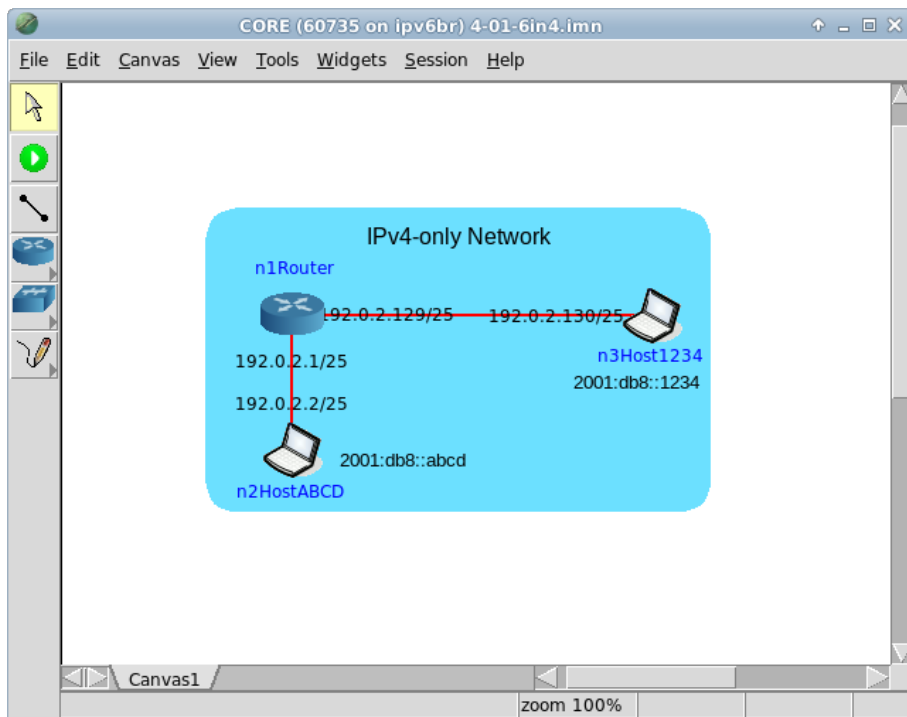


Figura 4.2: topologia da *Experiência 4.1* no CORE.

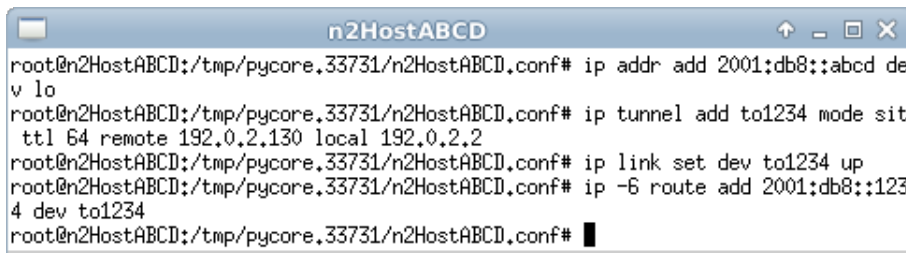
O objetivo desta topologia de rede é representar o mínimo necessário para que o túnel 6in4 seja entendido.

2. Conforme descrito nos Apêndices B e C, inicie a simulação e verifique a configuração de endereços em todos os nós. No caso do roteador, é possível observar que não há endereços IPv6 nas interfaces, nem mesmo endereços do tipo *link-local*, o que indica que o roteador não suporta o protocolo. Nos *hosts* n2HostABCD e n3Host1234, pode-se verificar a configuração tanto de endereços IPv4 quanto de endereços IPv6.

3. Ainda de acordo com o descrito no Apêndice C, verifique a conectividade tanto IPv4 quanto IPv6 entre todos os nós. Observe que há conectividade IPv4, mas ainda não há conectividade IPv6. O túnel a ser criado proverá essa conectividade.
 4. Configure o túnel `6in4`, entre `n2HostABCD` e `n3Host1234`.
- (a) Abra um terminal de `n2HostABCD` com um duplo-clique e utilize os seguintes comandos para a configuração:

```
# ip addr add 2001:db8::abcd dev lo
# ip tunnel add to1234 mode sit ttl 64 remote 192.0.2.130
  local 192.0.2.2
# ip link set dev to1234 up
# ip -6 route add 2001:db8::1234 dev to1234
```

O resultado dos comandos é representado pela Figura 4.3.

A terminal window titled "n2HostABCD" with standard window controls (minimize, maximize, close) in the top right. The terminal shows the following commands and their output:

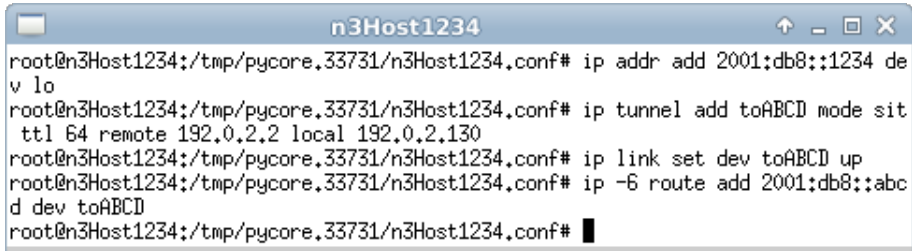
```
root@n2HostABCD:/tmp/pycore.33731/n2HostABCD.conf# ip addr add 2001:db8::abcd dev lo
root@n2HostABCD:/tmp/pycore.33731/n2HostABCD.conf# ip tunnel add to1234 mode sit
ttl 64 remote 192.0.2.130 local 192.0.2.2
root@n2HostABCD:/tmp/pycore.33731/n2HostABCD.conf# ip link set dev to1234 up
root@n2HostABCD:/tmp/pycore.33731/n2HostABCD.conf# ip -6 route add 2001:db8::1234 dev to1234
root@n2HostABCD:/tmp/pycore.33731/n2HostABCD.conf# █
```

Figura 4.3: resultado da configuração do túnel `6in4` em `n2HostABCD`.

- (b) Abra um terminal de `n3Host1234` com um duplo-clique e utilize os seguintes comandos para a configuração:

```
# ip addr add 2001:db8::1234 dev lo
# ip tunnel add toABCD mode sit ttl 64 remote 192.0.2.2
  local 192.0.2.130
# ip link set dev toABCD up
# ip -6 route add 2001:db8::abcd dev toABCD
```

O resultado dos comandos é representado pela Figura 4.4.



```

n3Host1234
root@n3Host1234:/tmp/pycore.33731/n3Host1234.conf# ip addr add 2001:db8::1234 de
v lo
root@n3Host1234:/tmp/pycore.33731/n3Host1234.conf# ip tunnel add toABCD mode sit
ttl 64 remote 192.0.2.2 local 192.0.2.130
root@n3Host1234:/tmp/pycore.33731/n3Host1234.conf# ip link set dev toABCD up
root@n3Host1234:/tmp/pycore.33731/n3Host1234.conf# ip -6 route add 2001:db8::abc
d dev toABCD
root@n3Host1234:/tmp/pycore.33731/n3Host1234.conf# █

```

Figura 4.4: resultado da configuração do túnel 6in4 em n3Host1234.

Observe que, em cada um dos nós, o túnel foi criado especificando-se o nome de interfaces de rede virtuais: toABCD e to1234; o tipo de túnel: sit, que é o nome utilizado pelo Linux para identificar o encapsulamento 6in4. Especificou-se também os endereços de origem e destino IPv4. Além disso, foi criada uma rota estática para a outra rede, que aponta para a interface virtual criada para o túnel. Pode-se ainda utilizar comandos como: ip addr show e ip -6 route show para verificar se a interface de túnel foi criada e se as rotas foram estabelecidas.

5. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface eth0 de n2HostABCD. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - (b) A verificação de conectividade IPv6 entre n2HostABCD e n3Host1234.
6. Efetue a análise dos pacotes coletados. Aplique o filtro icmpv6 no Wireshark para visualizar somente os pacotes que se quer observar, conforme exemplificado pela Figura 4.5.

Note que as versões mais atuais do Wireshark são inteligentes o suficiente para mostrar os pacotes como do tipo ICMPv6, mesmo que eles estejam encapsulados em pacotes IPv4.

Analise os pacotes *echo request* e *echo reply* e veja se os dados contidos neles conferem com a teoria, prestando atenção à forma como os pacotes IPv6 foram encapsulados em pacotes IPv4.

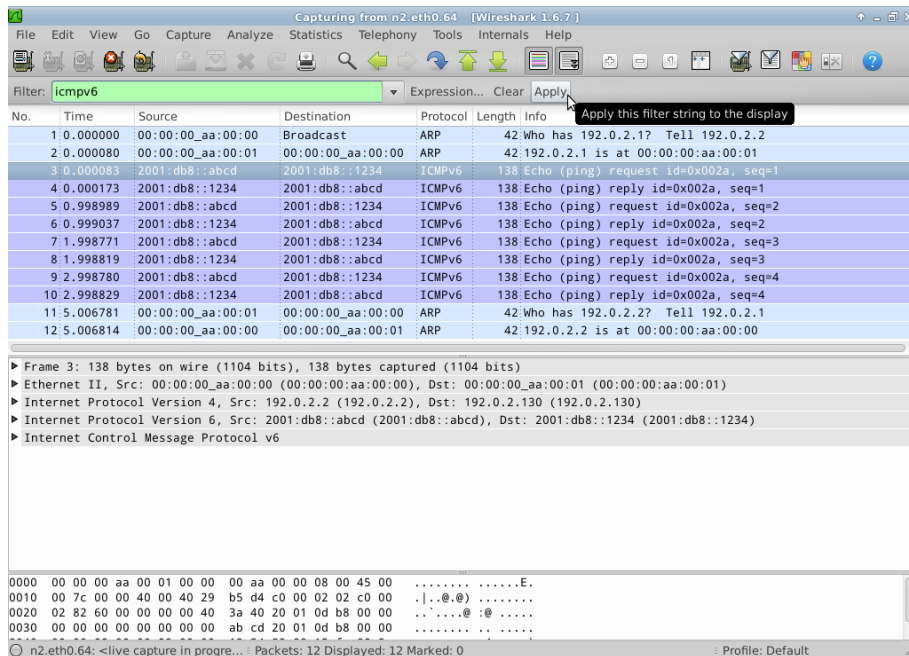


Figura 4.5: aplicação do filtro icmpv6 no Wireshark.

Campos importantes do pacote, representado pela Figura 4.6:

Type (camada Ethernet)

Indica que a mensagem utiliza IPv4.

Protocol (camada IPv4)

Indica que a mensagem encapsula um pacote IPv6 (protocolo número 41 – 6in4).

Source (camada IPv4)

A origem é o endereço IPv4 de n2HostABCD (192.0.2.2).

Destination (camada IPv4)

O destino é o endereço IPv4 de n3Host1234 (192.0.2.130).

Source (camada IPv6)

A origem é o endereço IPv6 de n2HostABCD (2001:db8::abcd).

Destination (camada IPv6)

O destino é o endereço IPv6 de n3Host1234 (2001:db8::1234).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 128 (*echo request*).

```

3 0.000083 2001:db8::abcd 2001:db8::1234 [ICMPv6 138 Echo (ping) request id=0x002a, seq=1]
Frame 3: 138 bytes on wire (1104 bits), 138 bytes captured (1104 bits) on interface 0
Ethernet II, Src: 00:00:00_aa:00:00 (00:00:00:aa:00:00), Dst: 00:00:00_aa:00:01 (00:00:00:aa:00:01)
Internet Protocol Version 4, Src: 192.0.2.2 (192.0.2.2), Dst: 192.0.2.130 (192.0.2.130)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 124
  Identification: 0x0000 (0)
  Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 64
  Protocol: IPv6 (41)
  Header checksum: 0xb5d4 [correct]
  Source: 192.0.2.2 (192.0.2.2)
  Destination: 192.0.2.130 (192.0.2.130)
Internet Protocol Version 6, Src: 2001:db8::abcd (2001:db8::abcd), Dst: 2001:db8::1234 (2001:db8::1234)
  0110 .... = Version: 6
  .... 0000 0000 .... = Traffic class: 0x00000000
  .... 0000 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
  Payload length: 64
  Next header: ICMPv6 (0x3a)
  Hop limit: 64
  Source: 2001:db8::abcd (2001:db8::abcd)
  Destination: 2001:db8::1234 (2001:db8::1234)
Internet Control Message Protocol v6
  Type: Echo (ping) request (128)
  Code: 0
  Checksum: 0x15fe [correct]
  Identifier: 0x002a
  Sequence: 1
  [Response In: 41]
  Data (56 bytes)
0010 00 7c 00 00 40 00 40 25 b5 d4 c0 00 02 02 c0 00 .|. @. @ .....
0020 02 82 60 00 00 00 00 40 3a 40 20 01 0d b8 00 00 ..... @ .....
0030 00 00 00 00 00 00 00 00 ab cd 20 01 0d b8 00 00 ..... . .....
0040 00 00 00 00 00 00 00 00 12 34 80 00 15 fe 00 2a ..... .4.....*

```

Figura 4.6: análise de pacote ICMPv6 echo request no Wireshark.

7. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 4.2. Túnel GRE

Objetivo

Esta experiência mostra o funcionamento do túnel GRE transportando IPv6 em uma rede que só funciona com IPv4.

O cenário utilizado tem dois nós pilha dupla e um roteador que funciona somente com IPv4. Um túnel GRE é configurado entre os dois nós e observa-se como os pacotes IPv6 são encapsulados e transportados por meio do roteador IPv4.

Para o presente exercício será utilizada a topologia descrita no arquivo: **4-02-GRE.imn**.

Introdução teórica

GRE (*Generic Routing Encapsulation*) é um tipo de encapsulamento genérico descrito na RFC 2784 (Farinacci *et al.*, 2000), atualizada pela RFC 2890 (Dommetty, 2000). O GRE tem um cabeçalho próprio. Ele pode transportar diversos tipos de protocolos ou ser transportado em vários tipos de protocolos.

Para encapsular pacotes IPv6 em IPv4, primeiramente acrescenta-se o cabeçalho GRE. O cabeçalho IPv4 é acrescentado depois e no campo Protocolo especifica-se o valor 47 (2F em hexadecimal), indicando que o IPv4 está transportando o GRE como *payload*. O túnel GRE é configurado estaticamente. A Figura 4.7 ilustra isso.

A vantagem do túnel GRE em relação ao 6in4 é que o primeiro pode transportar diversos protocolos simultaneamente, enquanto o segundo só transporta IPv6. Com GRE é possível, por exemplo, criar um túnel para transportar IPv6 e CLNS, usado pelo ISIS, simultaneamente. A desvantagem é o *overhead* maior.

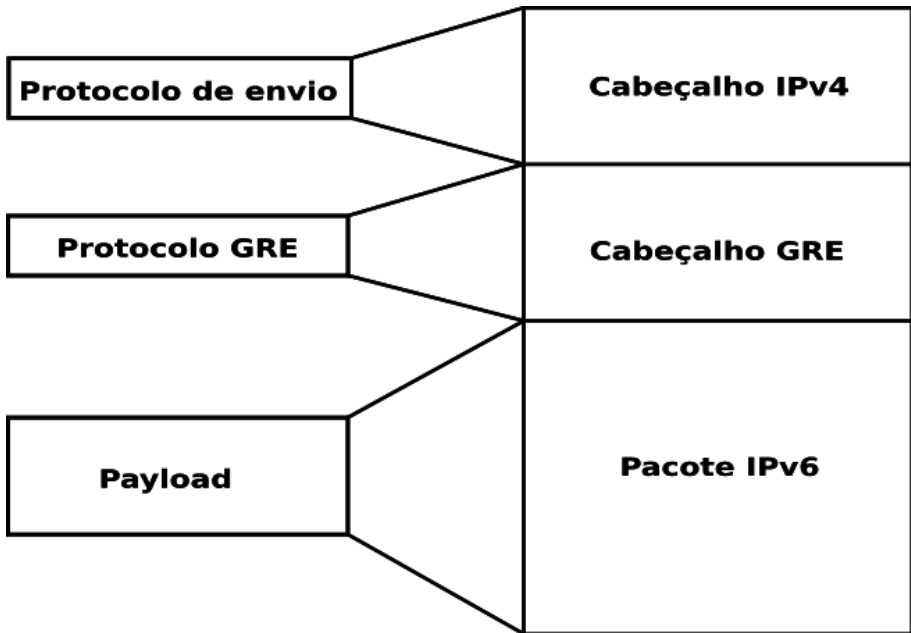


Figura 4.7: encapsulamento de pacote IPv6 em IPv4 utilizando GRE.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **4-02-GRE.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 4.8, deve aparecer.

O objetivo desta topologia de rede é representar o mínimo necessário para que o túnel GRE seja entendido.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação e verifique a configuração de endereços em todos os nós. No caso do roteador, pode-se observar que não há endereços IPv6 nas interfaces, nem mesmo endereços do tipo *link-local*, o que indica que o roteador não suporta o protocolo. Já nos nós n2HostABCD e n3Host1234, podemos verificar a configuração tanto de endereços IPv4 quanto de endereços IPv6.
3. Ainda de acordo com o descrito no Apêndice C, verifique a conectividade tanto IPv4 quanto IPv6 entre todos os nós. Observe que há conectividade IPv4, mas ainda não há conectividade IPv6. O túnel a ser criado proverá esta conectividade.

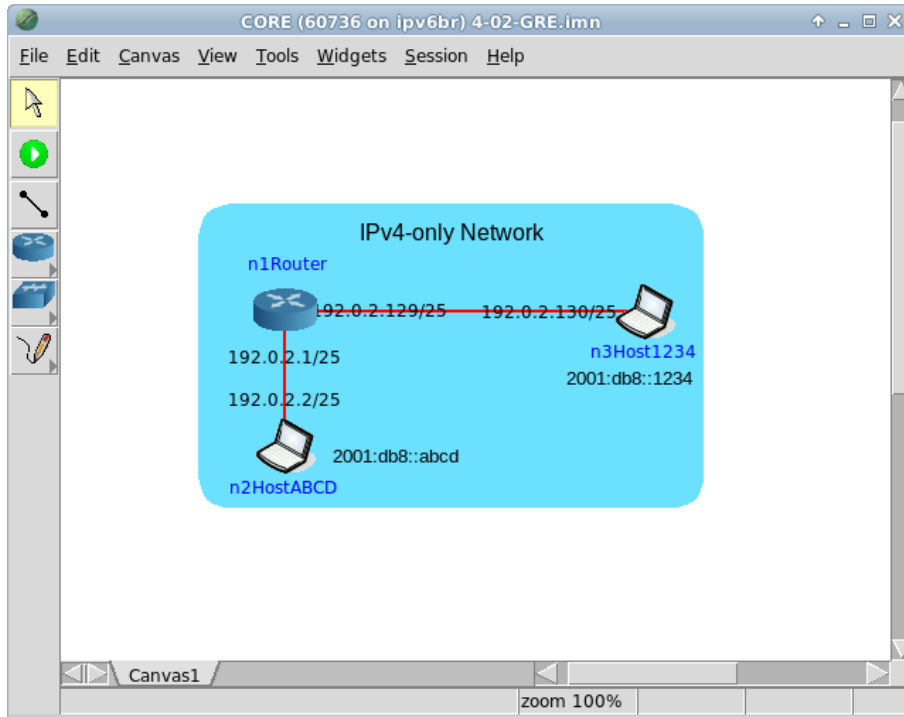


Figura 4.8: topologia da *Experiência 4.2* no CORE.

4. Configure o túnel GRE entre n2HostABCD e n3Host1234.
 - (a) Abra um terminal de n2HostABCD com um duplo-clique e utilize os seguintes comandos para a configuração:

```
# ip addr add 2001:db8::abcd dev lo
# ip tunnel add to1234 mode gre ttl 64 remote 192.0.2.130
  local 192.0.2.2
# ip link set dev to1234 up
# ip -6 route add 2001:db8::1234 dev to1234
```

O resultado dos comandos é representado pela Figura 4.9.

- (b) Abra um terminal de n3Host1234 com um duplo-clique e utilize os seguintes comandos para a configuração:

```
# ip addr add 2001:db8::1234 dev lo
# ip tunnel add toABCD mode gre ttl 64 remote 192.0.2.2
  local 192.0.2.130
```



```

root@n2HostABCD:/tmp/pycore.33738/n2HostABCD.conf# ip addr add 2001:db8::abcd de
v lo
root@n2HostABCD:/tmp/pycore.33738/n2HostABCD.conf# ip tunnel add to1234 mode gre
ttl 64 remote 192.0.2.130 local 192.0.2.2
root@n2HostABCD:/tmp/pycore.33738/n2HostABCD.conf# ip link set dev to1234 up
root@n2HostABCD:/tmp/pycore.33738/n2HostABCD.conf# ip -6 route add 2001:db8::123
4 dev to1234
root@n2HostABCD:/tmp/pycore.33738/n2HostABCD.conf# █

```

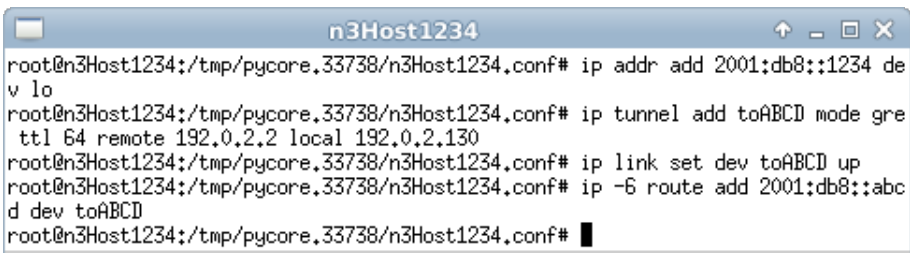
Figura 4.9: resultado da configuração do túnel GRE em n2HostABCD.

```

# ip link set dev toABCD up
# ip -6 route add 2001:db8::abcd dev toABCD

```

O resultado dos comandos é representado pela Figura 4.10.



```

root@n3Host1234:/tmp/pycore.33738/n3Host1234.conf# ip addr add 2001:db8::1234 de
v lo
root@n3Host1234:/tmp/pycore.33738/n3Host1234.conf# ip tunnel add toABCD mode gre
ttl 64 remote 192.0.2.2 local 192.0.2.130
root@n3Host1234:/tmp/pycore.33738/n3Host1234.conf# ip link set dev toABCD up
root@n3Host1234:/tmp/pycore.33738/n3Host1234.conf# ip -6 route add 2001:db8::abc
d dev toABCD
root@n3Host1234:/tmp/pycore.33738/n3Host1234.conf# █

```

Figura 4.10: resultado da configuração do túnel GRE em n3Host1234.

Observe que, em cada um dos nós o túnel foi criado especificando-se o nome de interfaces de rede virtuais: toABCD e to1234; o tipo de túnel: gre, que é o nome utilizado pelo Linux para identificar o encapsulamento utilizando este protocolo. Especificou-se também os endereços de origem e destino IPv4. Além disso, foi criada uma rota estática para a outra rede, que aponta para a interface virtual criada para o túnel. Pode-se ainda utilizar comandos como: ip addr show e ip -6 route show para verificar que a interface de túnel foi criada e que as rotas foram estabelecidas.

5. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface eth0 de n2HostABCD. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - (b) A verificação de conectividade IPv6 entre n2HostABCD e n3Host1234.

6. Efetue a análise dos pacotes coletados. Aplique o filtro icmpv6 no Wireshark para visualizar somente os pacotes que se quer observar, conforme exemplificado pela Figura 4.11.

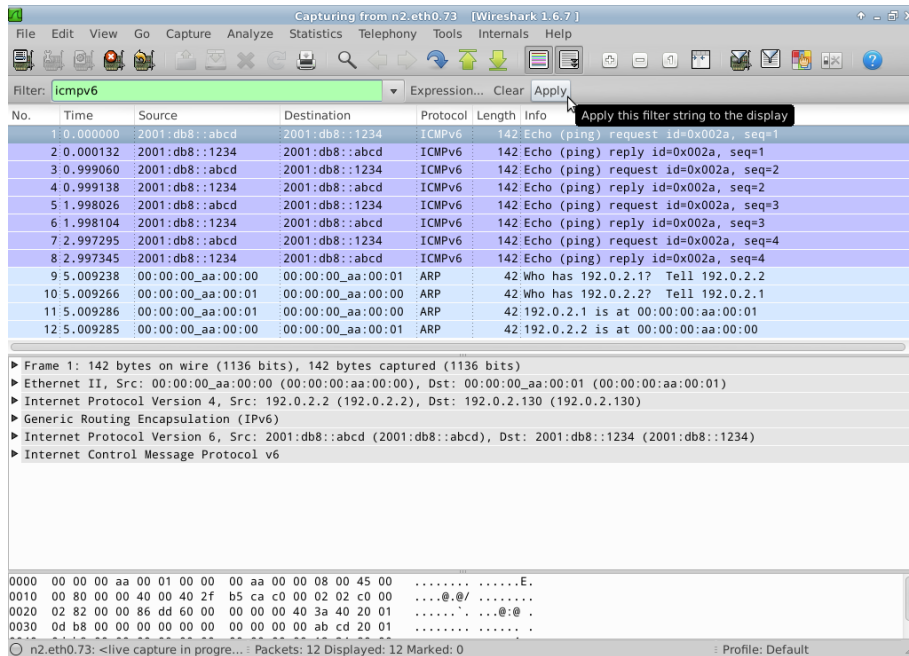


Figura 4.11: aplicação do filtro icmpv6 no Wireshark.

Note que as versões mais atuais do Wireshark são inteligentes o suficiente para mostrar pacotes do tipo ICMPv6, mesmo eles estando encapsulados em pacotes IPv4, utilizando GRE.

Analise os pacotes *echo request* e *echo reply*. Veja se os dados contidos neles conferem com a teoria, prestando atenção à forma com que os pacotes IPv6 foram encapsulados em pacotes IPv4, usando GRE.

Campos importantes do pacote, representado pela Figura 4.12:

Type (camada Ethernet)

Indica que a mensagem utiliza IPv4.

Source (camada IPv4)

A origem é o endereço IPv4 de umaPonta (192.0.2.2).

Destination (camada IPv4)

O destino é o endereço IPv4 de outraPonta (192.0.2.130).

Protocol Type (camada GRE)

Indica que a mensagem encapsula um pacote IPv6.

Source (camada IPv6)

A origem é o endereço IPv6 de outraPonta (2001:db8::abcd).

Destination (camada IPv6)

O destino é o endereço IPv6 de umaPonta (2001:db8::1234).

Type (camada ICMPv6)

Indica que a mensagem é do tipo 128 (*echo request*).

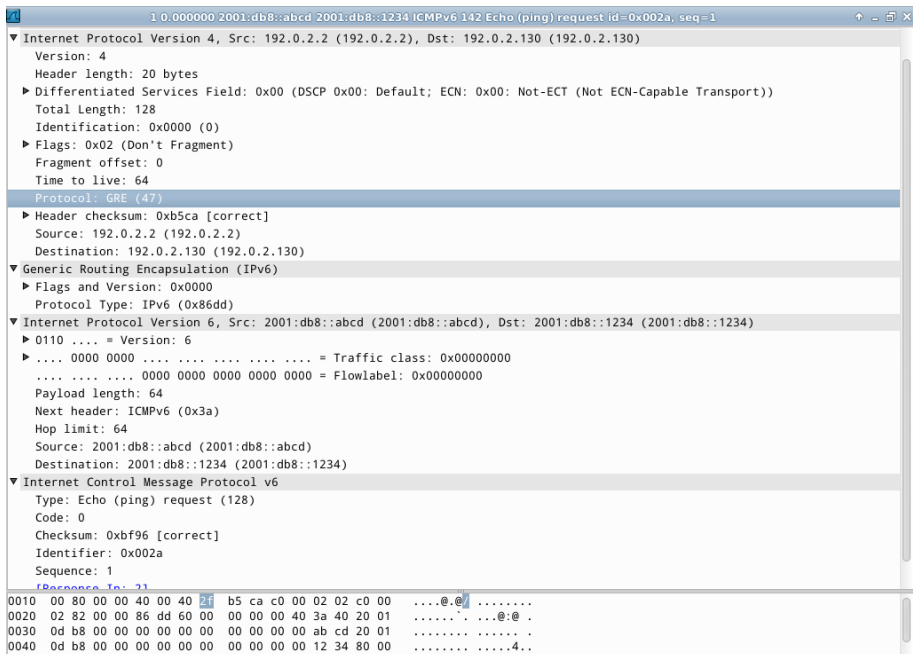


Figura 4.12: análise de pacote ICMPv6 echo request no Wireshark.

7. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 4.3. Dual Stack Lite (DS-Lite): implantação

Objetivo

O objetivo desta experiência é mostrar o funcionamento da técnica de transição DS-Lite, que permite ao provedor entregar IPs legados (IPv4) e compartilhados para os usuários, sobre túneis automáticos, numa rede IPv6.

Na rede do provedor será configurado um roteador, chamado de AFTR, e na rede do cliente será configurado um dispositivo chamado de B4. Essas configurações permitirão a estes equipamentos estabelecerem um túnel entre si e fornecerem conectividade IPv4 à rede do cliente.

Este laboratório utilizará a topologia descrita no arquivo: **4-03-DS-Lite.imn**.

Introdução teórica

O *Dual Stack Lite* é uma técnica padronizada pela RFC 6333 (Durand *et al.*, 2011). Ela pode ser aplicada em situações em que o provedor já oferece IPv6 nativo para seus usuários.

Sua implantação necessita de um equipamento denominado AFTR (*Address Family Transition Router*), que implementa um CGN (*Carrier Grade NAT*), um NAT de grande porte, na rede do provedor. Entre o AFTR e cada CPE (*Customer Premise Equipment*) de usuário, utiliza-se um túnel IPv4 sobre IPv6 para transportar o tráfego IPv4. No contexto do DS-Lite, o CPE do usuário é chamado B4 (*DS-Lite Basic Bridging BroadBand*). Nas extremidades desses túneis são usados endereços da faixa 192.0.0.0/29, especialmente reservada para este fim. Para o CPE do usuário e os demais equipamentos da rede do usuário, são utilizados IPs da RFC 1918 (Rekhter *et al.*, 1996).

Não há problema se diferentes usuários utilizarem faixas de IPs repetidas, pois o AFTR identifica os diferentes túneis com base no IPv6 de origem dos pacotes encapsulados.

É importante frisar alguns pontos:

1. O AFTR usa CGN, mas não força o usuário a utilizar duplo NAT. Ou seja, o AFTR realiza a função de NAT, de forma concentrada, para cada um dos dispositivos de cada usuário.
2. O DS-Lite utiliza endereços privados na faixa 192.0.0.0/29 para as extremidades dos túneis IPv4 sobre IPv6, evitando a utilização desnecessária de endereços IPv4 na infraestrutura do provedor.

A Figura 4.13 exemplifica o funcionamento do DS-Lite.

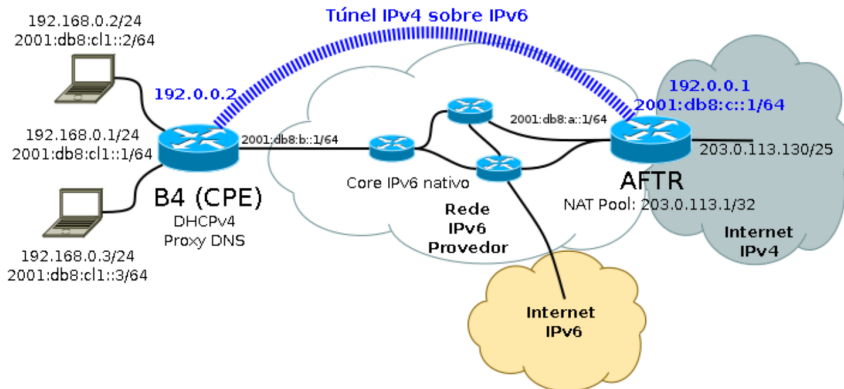


Figura 4.13: topologia da técnica de transição DS-Lite.

Roteiro experimental

1. A principal meta desta experiência é a simulação da implantação da técnica DS-Lite na rede de um ISP (*Internet Service Provider*), fazendo com que o aluno comece a se familiarizar com suas funcionalidades.
2. Inicie o CORE e abra o arquivo **4-03-DS-Lite.imn** localizado no diretório lab, dentro do Desktop. A topologia representada pela Figura 4.14 deverá ser exibida.

Essa topologia ilustra a situação em que um determinado ISP possui uma rede unicamente IPv6 funcional e deseja implantar a técnica DS-Lite para que seus clientes passem a ter acesso à Internet IPv4.

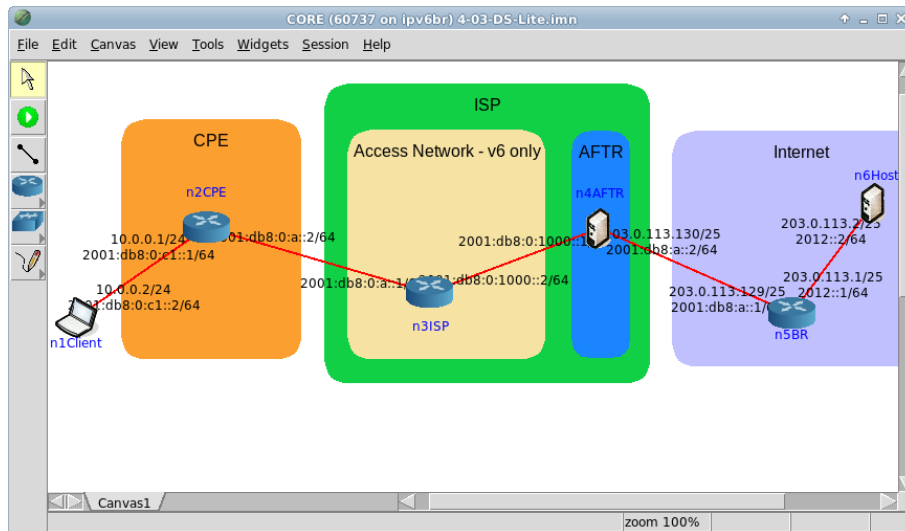


Figura 4.14: topologia da *Experiência 4.3* no CORE.

Na situação inicial da simulação, a rede foi configurada com rotas estáticas, de forma que todas as máquinas pudessem se conectar por meio do IPv6. Porém, na prática, o provedor pode utilizar quaisquer outras técnicas para distribuir IPv6 a sua infraestrutura e aos seus clientes.

3. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós n1Client e n6Host e a conectividade entre eles.
4. A partir desse passo será iniciada a implantação da técnica de transição DS Lite na rede simulada. O servidor de borda do ISP será configurado como um roteador AFTR, que irá realizar a tarefa de NATv4 sobre a rede IPv6 do servidor. Enquanto isso, o CPE será configurado como um B4. Isto é necessário para fechar um túnel que encapsule os pacotes IPv4 na rede IPv6 do ISP. A Figura 4.15 exemplifica esta situação.
5. A configuração será iniciada pelo CPE. Acesse o terminal da máquina n2CPE com um duplo clique sobre ela e execute os seguintes comandos:

```
# modprobe ip6_tunnel
# ip -6 tunnel add dsltun mode ipip6 remote
    2001:db8:0:2000:: local 2001:db8:0:a::2 dev eth1
```

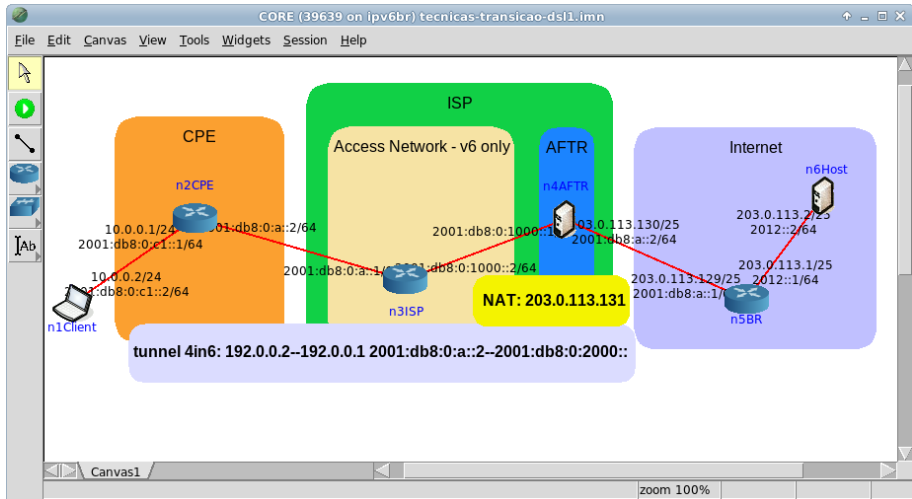


Figura 4.15: topologia da experiência após configurações iniciais.

```
# ip addr add 192.0.0.2 peer 192.0.0.1 dev dsltn
# ip link set dev dsltn up
# ip route add default dev dsltn
```

O terminal ficará como o representado na Figura 4.16.

```
n2CPE
root@n2CPE:/tmp/pycore.54893/n2CPE.conf# modprobe ip6_tunnel
root@n2CPE:/tmp/pycore.54893/n2CPE.conf# ip -6 tunnel add dsltn mode ipip6 remo
te 2001:db8:0:2000:: local 2001:db8:0:a::2 dev eth1
root@n2CPE:/tmp/pycore.54893/n2CPE.conf# ip addr add 192.0.0.2 peer 192.0.0.1 de
v dsltn
root@n2CPE:/tmp/pycore.54893/n2CPE.conf# ip link set dev dsltn up
root@n2CPE:/tmp/pycore.54893/n2CPE.conf# ip route add default dev dsltn
root@n2CPE:/tmp/pycore.54893/n2CPE.conf# █
```

Figura 4.16: comandos de configuração do DS-Lite para o nó CPE.

Esta sequência de comandos cria uma nova interface de rede virtual chamada `dsltn`, que encapsula todos os pacotes IPv4 vindos da rede local do cliente em pacotes IPv6 e os envia para o endereço `2001:db8:0:2000::`, que será configurado como ponta de saída do túnel na máquina AFTR.

6. Em paralelo, efetue:

- (a) A coleta dos pacotes trafegados na interface eth1 de n4AFTR. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - (b) A verificação de conectividade IPv4 entre n1Client e n6Host.
7. Efetue a análise dos pacotes coletados. Verifique os pacotes capturados e note que, pela interface capturada, passam pacotes ICMPv4 da origem 10.0.0.2 para o destino 203.0.113.2, encapsulados com cabeçalhos IPv6. Note que, de fato, houve 100% de perda de pacotes em razão da ponta de saída do túnel ainda não ter sido configurada.
 8. Com o túnel 4in6 funcionando por parte do CPE, configure a máquina n4AFTR para fechar o túnel e realizar a função de NAT.

Abra o terminal da máquina n4AFTR e siga os passos:

- (a) No terminal dessa mesma máquina, verifique a existência de um arquivo denominado aftr-script. Seu conteúdo deve se apresentar da seguinte forma:

```
#!/bin/sh
aftr_start() {
    set -x
    ip link set tun0 up
    ip addr add 192.0.0.1 peer 192.0.0.2 dev tun0
    ip route add 203.0.113.131/32 dev tun0
    ip -6 addr add fe80::1 dev tun0
    ip -6 route add 2001:db8:0:2000::/64 dev tun0
    arp -i eth0 -s 203.0.113.131 0a:0b:0c:0d:0e:f0 pub
}
aftr_stop() {
    set -x
    ip link set tun0 down
}
case "$1" in
start)
    aftr_start
;;
```

```
stop)
    aftr_stop
    ;;
*)
    echo "Usage: $0 start|stop"
    exit 1
    ;;
esac
exit 0
```

Neste arquivo é importante notar a função dos endereços.

192.0.0.1 e 192.0.0.2

São especificamente designados pela RFC 6333 (Durand *et al.*, 2011) para a configuração das interfaces de túnel nas implantações do DS-Lite.

203.0.113.131

É o endereço público utilizado para a realização do NAT na rede interna do ISP. Ele deve ser diferente do endereço utilizado na interface física do servidor n4AFTR.

2001:db8:0:2000::

É um endereço arbitrariamente escolhido para fechar o túnel no servidor n4AFTR. Este endereço não pode ser utilizado nas interfaces de rede do servidor ou por qualquer outro equipamento na rede.

- (b) Verifique também a existência de um arquivo denominado `aftr.conf`. Seu conteúdo deve estar da seguinte forma:

```
default tunnel mss on
defmtu 1450
address endpoint 2001:db8:0:2000::
address icmp 203.0.113.131
pool 203.0.113.131
acl6 ::0/0
```

Novamente:

203.0.113.131

É o endereço público utilizado para a realização do NAT na rede interna do ISP. Ele deve ser diferente do endereço utilizado na interface física do servidor AFTR.

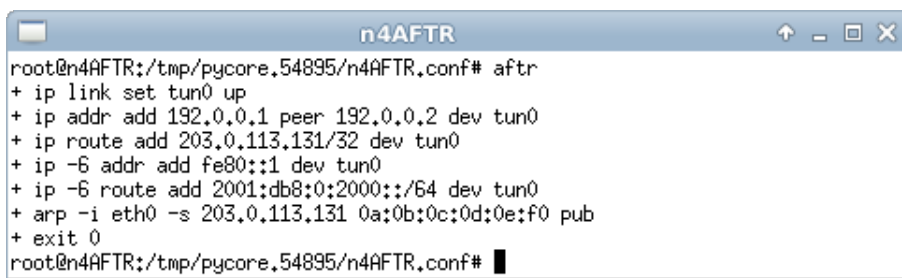
2001:db8:0:2000::

É um endereço arbitrariamente escolhido para fechar o túnel no servidor AFTR. Esse endereço não pode ser utilizado nas interfaces de rede do servidor ou por qualquer outro equipamento na rede.

- (c) Inicie o serviço `aftr`:

```
# aftr
```

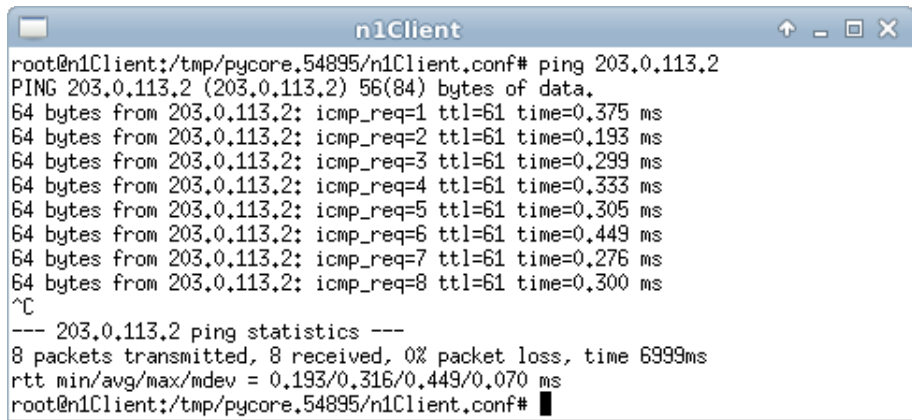
O resultado do comando é representado pela Figura 4.17.



```
n4AFTR
root@n4AFTR:/tmp/pycore,.54895/n4AFTR.conf# aftr
+ ip link set tun0 up
+ ip addr add 192.0.0.1 peer 192.0.0.2 dev tun0
+ ip route add 203.0.113.131/32 dev tun0
+ ip -6 addr add fe80::1 dev tun0
+ ip -6 route add 2001:db8:0:2000::/64 dev tun0
+ arp -i eth0 -s 203.0.113.131 0a:0b:0c:0d:0e:f0 pub
+ exit 0
root@n4AFTR:/tmp/pycore,.54895/n4AFTR.conf# █
```

Figura 4.17: comando de inicialização do DS-Lite para o nó `n4AFTR`.

9. Com isso implantação da técnica está finalizada. Para testar seu funcionamento, utilize o comando `ping 203.0.113.2` a partir da máquina `n1Client` para verificar se ela possui conectividade IPv4 com a Internet. A saída deve ser similar à representada na Figura 4.18.

A terminal window titled 'n1Client' showing the execution of a ping command to the IP address 203.0.113.2. The terminal displays the command prompt, the ping command, and the resulting output for 8 requests. The output shows that all 8 packets were received with 0% packet loss. The round-trip times (rtt) are listed as min/avg/max/mdev = 0,193/0,316/0,449/0,070 ms. The terminal prompt is root@n1Client:~/tmp/pycore.54895/n1Client.conf#.

```
root@n1Client:~/tmp/pycore.54895/n1Client.conf# ping 203.0.113.2
PING 203.0.113.2 (203.0.113.2) 56(84) bytes of data:
64 bytes from 203.0.113.2: icmp_req=1 ttl=61 time=0.375 ms
64 bytes from 203.0.113.2: icmp_req=2 ttl=61 time=0.193 ms
64 bytes from 203.0.113.2: icmp_req=3 ttl=61 time=0.299 ms
64 bytes from 203.0.113.2: icmp_req=4 ttl=61 time=0.333 ms
64 bytes from 203.0.113.2: icmp_req=5 ttl=61 time=0.305 ms
64 bytes from 203.0.113.2: icmp_req=6 ttl=61 time=0.449 ms
64 bytes from 203.0.113.2: icmp_req=7 ttl=61 time=0.276 ms
64 bytes from 203.0.113.2: icmp_req=8 ttl=61 time=0.300 ms
^C
--- 203.0.113.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 6999ms
rtt min/avg/max/mdev = 0,193/0,316/0,449/0,070 ms
root@n1Client:~/tmp/pycore.54895/n1Client.conf# █
```

Figura 4.18: *conectividade do cliente com o servidor do DS-Lite após configurações.*

10. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 4.4. IPv6 Rapid Deployment (6rd): configuração 6rd no *relay* e em um CPE (/64)

Objetivo

Esta experiência demonstra o funcionamento da técnica *IPv6 Rapid Deployment (6rd)*, que permite oferecer conectividade IPv6 aos usuários através de um túnel IPv4. Para isto, será efetuada a configuração manual do *relay*, na rede do provedor, e de um CPE, na rede do usuário, fornecendo uma rede /64.

Para o presente exercício, será utilizada a topologia descrita no arquivo: **4-04-6rd-one-CPE.imn**.

Introdução teórica

O 6rd permite que a infraestrutura da rede de acesso IPv4 seja utilizada sem modificações, para fazer uma implantação rápida do IPv6 até o usuário final. A técnica está descrita na RFC 5569 (Despres, 2010). Há dois elementos principais: o CPE 6rd e o *Relay* 6rd. O CPE 6rd funciona como um CPE IPv4 normal e atribui um endereço legado ao usuário, mas atribui também um endereço IPv6, que é formado a partir do endereço IPv4 e de um prefixo fornecido pelo provedor. O *Relay* 6rd fica na rede do provedor e tem conectividade nativa IPv6 e IPv4. O encapsulamento é o 6in4.

A Figura 4.19 ilustra a forma como o endereço IPv6 é criado.

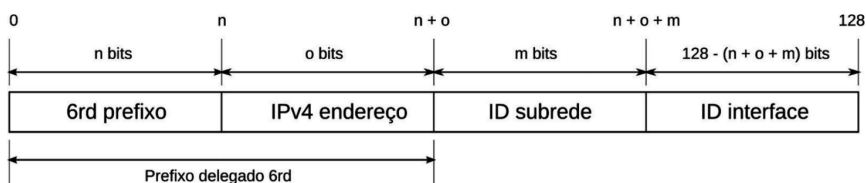


Figura 4.19: construção do endereço IPv6 para o cliente do 6rd (/64).

Deve-se notar que 6rd não resolve o problema da escassez de endereços IPv4. A técnica só pode ser usada quando há endereços disponíveis. Além disso, o 6rd exige a atualização dos *software* dos CPEs ou troca dos mesmos. Sua utilização é recomendada para provedores que não vão sofrer com o esgotamento de endereços em curto ou médio prazo. Por exemplo, porque sua base de usuários cresce muito pouco. Além disso, o provedor deve ter a gerência sobre os CPEs e ser capaz de fazer um *upgrade* para suportar a técnica.

No 6rd, o tamanho n do prefixo e o tamanho o do endereço IPv4, que formam o prefixo delegado 6rd, são escolhas do provedor de acesso. Para permitir que a autoconfiguração de endereço *stateless* funcione é necessário que o tamanho deste prefixo $n + o$ seja menor ou igual a 64 *bits*.

Normalmente utiliza-se $n=32$, $o=32$ e $m=0$. Pode-se, contudo, aumentar o número de *bits* utilizados por n para além de 32 e fazer com que o endereço IPv4 ocupe menos que 32 *bits*. Tal configuração é possível se os endereços IPv4 atribuídos a diferentes clientes fizerem parte de uma mesma rede, pois pode-se omitir o prefixo da rede. Por exemplo, se todos os endereços IPv4 forem da rede 198.51.0.0/16, os 16 *bits* que representam os números 198 e 51 podem ser omitidos e a representação do endereço IPv4 necessitará somente de 16 *bits*, ao contrário dos 32 *bits* necessários para representar o endereço completo.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **4-04-6rd-one-CPE.imn** localizado no diretório lab, dentro do Desktop. A topologia da rede, representada pela Figura 4.20, deve aparecer.

O objetivo dessa topologia de rede é representar o mínimo necessário para que a implantação do 6rd seja entendida. O roteador n3Router funciona apenas com IPv4. A rede foi configurada com rotas estáticas de forma que todas as máquinas possam conectar-se por meio do IPv4. O *relay* n4Relay e o servidor n5Host também possuem conectividade IPv6.

Neste experimento, o cliente receberá uma rede /64. O prefixo é composto pelo prefixo IPv6 do ISP e o endereço IPv4 do CPE.

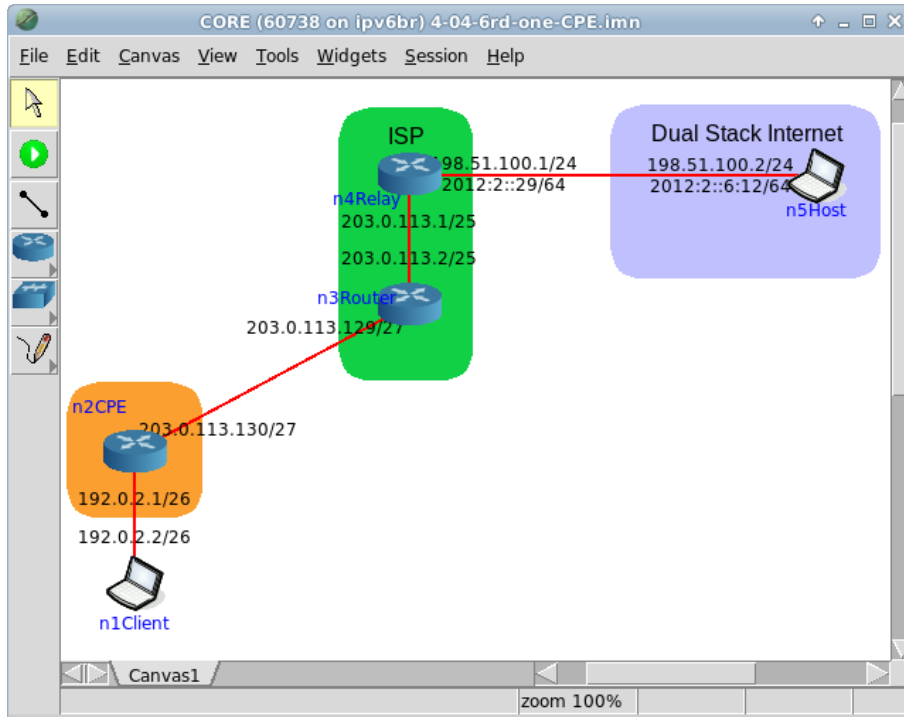


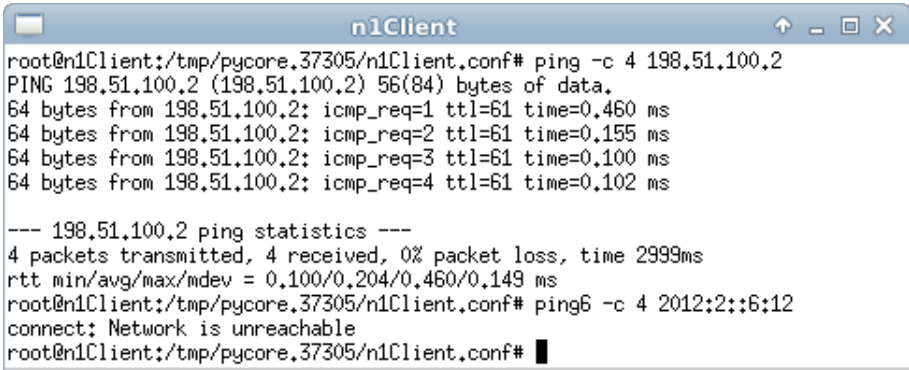
Figura 4.20: topologia da *Experiência 4.4* no CORE.

2. Conforme descrito nos Apêndices B e C, inicie a simulação, verifique a configuração de endereços IPv6 e IPv4 nos nós n1Client, n2CPE, n3Router, n4Relay e n5Host.
3. Veja que o roteador n3Router não suporta IPv6. Não há endereços IPv6 nas interfaces, nem mesmo endereços do tipo *link-local*, o que indica que o roteador não suporta o protocolo.
4. Verifique a conectividade entre n1Client e n5Host.
 - (a) Abra o terminal de n1Client, com um duplo-clique.
 - (b) Utilize os seguintes comandos para verificar a conectividade:

```
# ping -c 4 198.51.100.2
# ping6 -c 4 2012::6:12
```

O resultado do comando é representado pela Figura 4.21.

Observe que há conectividade IPv4, mas não há ainda conectividade IPv6. A configuração de 6rd proverá essa conectividade.



```

root@n1Client:/tmp/pycore.37305/n1Client.conf# ping -c 4 198.51.100.2
PING 198.51.100.2 (198.51.100.2) 56(84) bytes of data.
64 bytes from 198.51.100.2: icmp_req=1 ttl=61 time=0.460 ms
64 bytes from 198.51.100.2: icmp_req=2 ttl=61 time=0.155 ms
64 bytes from 198.51.100.2: icmp_req=3 ttl=61 time=0.100 ms
64 bytes from 198.51.100.2: icmp_req=4 ttl=61 time=0.102 ms

--- 198.51.100.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.100/0.204/0.460/0.149 ms
root@n1Client:/tmp/pycore.37305/n1Client.conf# ping6 -c 4 2012:::6:12
connect: Network is unreachable
root@n1Client:/tmp/pycore.37305/n1Client.conf# █

```

Figura 4.21: teste de conectividade entre n1Client e nó n5Host de rede 6rd.

5. Configure o 6rd no n4Relay e no n2CPE e um endereço IPv6 no n1Client.
 - (a) Abra o terminal de n4Relay, com um duplo-clique, e utilize os seguintes comandos para a configuração:

```

# ip tunnel add toClient mode sit local 203.0.113.1 ttl 64
# ip tunnel 6rd dev toClient 6rd-prefix 2001:db8::/32
# ip link set toClient up
# ip -6 addr add 2001:db8::1/128 dev toClient
# ip -6 route add 2001:db8::/32 dev toClient
# ip -6 route add 2000::/3 dev eth1

```

O resultado do comando é representado na Figura 4.22.



```

root@n4Relay:/tmp/pycore.37305/n4Relay.conf# ip tunnel add toClient mode sit local 203.0.113.1 ttl 64
root@n4Relay:/tmp/pycore.37305/n4Relay.conf# ip tunnel 6rd dev toClient 6rd-prefix 2001:db8::/32
root@n4Relay:/tmp/pycore.37305/n4Relay.conf# ip link set toClient up
root@n4Relay:/tmp/pycore.37305/n4Relay.conf# ip -6 addr add 2001:db8::1/128 dev toClient
root@n4Relay:/tmp/pycore.37305/n4Relay.conf# ip -6 route add 2001:db8::/32 dev toClient
root@n4Relay:/tmp/pycore.37305/n4Relay.conf# ip -6 route add 2000::/3 dev eth1
root@n4Relay:/tmp/pycore.37305/n4Relay.conf# █

```

Figura 4.22: comandos de configuração do relay do 6rd (/64).

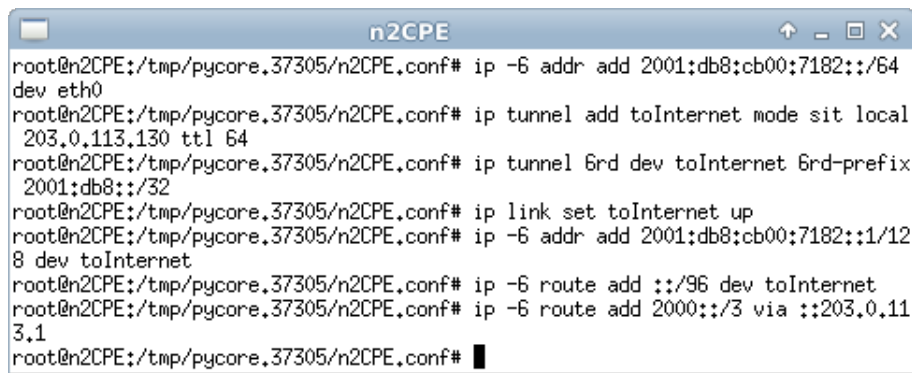
O n4Relay possui conectividade IPv4 e IPv6 para a Internet, aqui representada pelo nó n5Host, enquanto a rede interna está devidamente configurada somente para o uso de IPv4. A configuração anterior cria o túnel toClient, cujo nome refere-se à rede interna do ISP.

Foi associado ao túnel um endereço IPv4 da rede do ISP e habilitado o encapsulamento utilizando o protocolo 41, representado no Linux pelo tipo sit. O prefixo IPv6 do ISP, o 2001:db8::/32, foi informado como parâmetro de configuração do túnel 6rd pois será utilizado para compor o endereço IPv6 do cliente. Após inicializar o túnel, também foram configuradas duas rotas estáticas: uma para a rede interna através do túnel; e outra para a Internet, representada pela interface eth1.

- (b) Abra o terminal de n2CPE com um duplo-clique e utilize os seguintes comandos para a configuração:

```
# ip -6 addr add 2001:db8:cb00:7182::/64 dev eth0
# ip tunnel add toInternet mode sit local 203.0.113.130 ttl 64
# ip tunnel 6rd dev toInternet 6rd-prefix 2001:db8::/32
# ip link set toInternet up
# ip -6 addr add 2001:db8:cb00:7182::1/128 dev toInternet
# ip -6 route add ::/96 dev toInternet
# ip -6 route add 2000::/3 via ::203.0.113.1
```

O resultado do comando é representado na Figura 4.23.



```
root@n2CPE:/tmp/pycore.37305/n2CPE.conf# ip -6 addr add 2001:db8:cb00:7182::/64
dev eth0
root@n2CPE:/tmp/pycore.37305/n2CPE.conf# ip tunnel add toInternet mode sit local
203.0.113.130 ttl 64
root@n2CPE:/tmp/pycore.37305/n2CPE.conf# ip tunnel 6rd dev toInternet 6rd-prefix
2001:db8::/32
root@n2CPE:/tmp/pycore.37305/n2CPE.conf# ip link set toInternet up
root@n2CPE:/tmp/pycore.37305/n2CPE.conf# ip -6 addr add 2001:db8:cb00:7182::1/12
8 dev toInternet
root@n2CPE:/tmp/pycore.37305/n2CPE.conf# ip -6 route add ::/96 dev toInternet
root@n2CPE:/tmp/pycore.37305/n2CPE.conf# ip -6 route add 2000::/3 via ::203.0.11
3.1
root@n2CPE:/tmp/pycore.37305/n2CPE.conf# █
```

Figura 4.23: comandos de configuração do CPE do 6rd (/64).

Nesta configuração, é atribuído um endereço IPv6 à interface de rede local `eth0`. Ele é composto pelo prefixo IPv6 do ISP, o `2001:db8::/32`, e o endereço IPv4 do CPE convertido em hexadecimal, `203.0.113.130` para `0xcb 0x00 0x71 0x82`, resultando no prefixo `2001:db8:cb00:7182::/64`.

Em seguida, é configurado o túnel `toInternet`, cujo nome refere-se à rede interna do ISP utilizada para trafegar os pacotes destinados à Internet. Foi associado ao túnel um endereço IPv4 da rede do ISP e habilitado o encapsulamento utilizando o protocolo 41, representado no Linux pelo tipo `sit`.

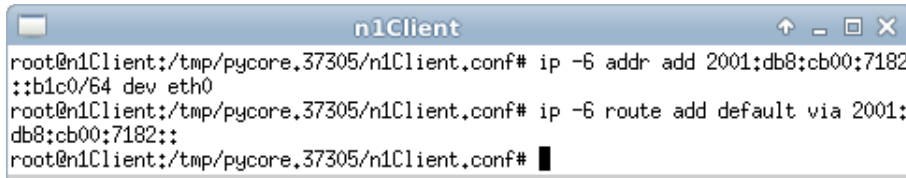
O prefixo IPv6 do ISP, o `2001:db8::/32`, foi informado como parâmetro de configuração do túnel `6rd`, pois será utilizado para compor o endereço IPv6 do cliente. Após inicializar o túnel, são configuradas duas rotas estáticas: uma para viabilizar o uso do túnel em IPv4, `::/96`; e outra para a Internet, por meio do encapsulamento para o *relay*, por meio do IPv4 `::203.0.113.1`.

Note que em um ambiente de produção, a configuração da CPE deve ser automática. Os parâmetros necessários podem ser informados por uma variedade de meios, como por uma opção no DHCP, um campo no DNS, SNMP, usando TR-069, etc.

- (c) Abra o terminal de n1Client com um duplo-clique e utilize os seguintes comandos para a configuração:

```
# ip -6 addr add 2001:db8:cb00:7182::b1c0/64 dev eth0
# ip -6 route add default via 2001:db8:cb00:7182::
```

O resultado do comando é representado na Figura 4.24.



```
n1Client
root@n1Client:/tmp/pycore.37305/n1Client.conf# ip -6 addr add 2001:db8:cb00:7182::b1c0/64 dev eth0
root@n1Client:/tmp/pycore.37305/n1Client.conf# ip -6 route add default via 2001:db8:cb00:7182::
root@n1Client:/tmp/pycore.37305/n1Client.conf# █
```

Figura 4.24: comandos de configuração do cliente do 6rd (/64).

Neste experimento, o endereço IPv6 do n1Client é configurado manualmente, pois seu objetivo é apenas o de demonstrar o funcionamento do 6rd. Em um ambiente de produção, o CPE deve ser configurado para prover a autoconfiguração para os clientes.

Observe que, neste experimento, os valores apresentados na introdução teórica de **n**, **o** e **m** são, respectivamente, 32, 32 e 0 *bits*, de modo que o maior prefixo IPv6 do ISP foi utilizado juntamente com o endereço IPv4 integral do CPE.

6. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface eth0 de n3Router. As instruções de coleta de pacotes utilizando tcpdump ou Wireshark se encontram no Apêndice C.
 - (b) Efetue os seguintes passos enquanto a coleta é realizada:
 - i. No terminal de n1Client, utilize o seguinte comando:

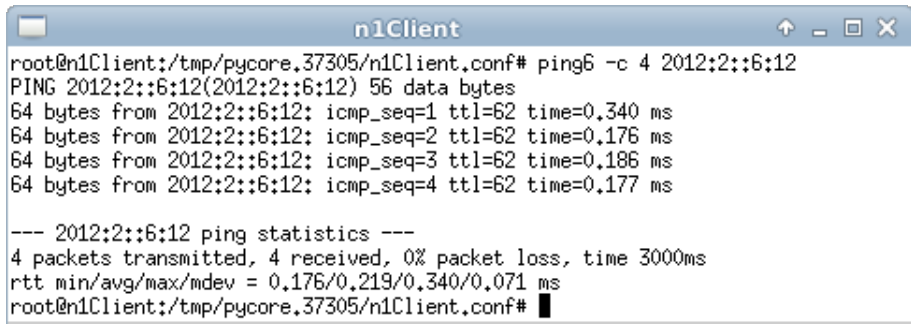
```
# ping6 -c 4 2012:2::6:12
```

O resultado do comando é representado na Figura 4.25.

7. No terminal de n1Client, utilize o seguinte comando:

```
# traceroute6 2012:2::6:12
```


O resultado do comando é representado na Figura 4.26.



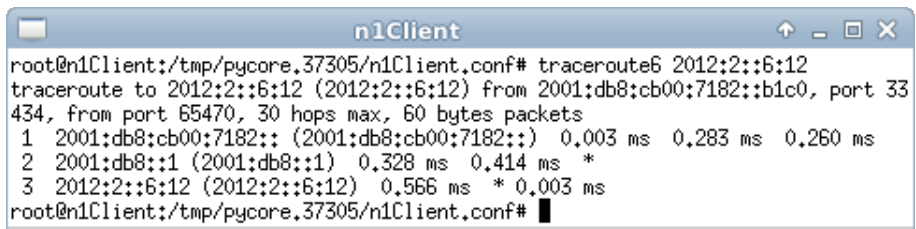
```

n1Client
root@n1Client:/tmp/pycore.37305/n1Client.conf# ping6 -c 4 2012:2::6:12
PING 2012:2::6:12(2012:2::6:12) 56 data bytes
64 bytes from 2012:2::6:12: icmp_seq=1 ttl=62 time=0,340 ms
64 bytes from 2012:2::6:12: icmp_seq=2 ttl=62 time=0,176 ms
64 bytes from 2012:2::6:12: icmp_seq=3 ttl=62 time=0,186 ms
64 bytes from 2012:2::6:12: icmp_seq=4 ttl=62 time=0,177 ms

--- 2012:2::6:12 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0,176/0,219/0,340/0,071 ms
root@n1Client:/tmp/pycore.37305/n1Client.conf# █

```

Figura 4.25: saída do ping6 no cliente do 6rd (/64).



```

n1Client
root@n1Client:/tmp/pycore.37305/n1Client.conf# traceroute6 2012:2::6:12
traceroute to 2012:2::6:12 (2012:2::6:12) from 2001:db8:cb00:7182::b1c0, port 33
434, from port 65470, 30 hops max, 60 bytes packets
 1  2001:db8:cb00:7182:: (2001:db8:cb00:7182::)  0,003 ms  0,283 ms  0,260 ms
 2  2001:db8::1 (2001:db8::1)  0,328 ms  0,414 ms  *
 3  2012:2::6:12 (2012:2::6:12)  0,566 ms  * 0,003 ms
root@n1Client:/tmp/pycore.37305/n1Client.conf# █

```

Figura 4.26: saída de traceroute6 no cliente do 6rd (/64).

8. Efetue a análise dos pacotes coletados. Aplique o filtro icmpv6 no Wireshark e procure pelo pacote *echo request* e *echo reply*. Analise-os e veja se os dados contidos nos pacotes conferem com o que foi passado na teoria, prestando atenção à forma com que os pacotes IPv6 foram encapsulados em pacotes IPv4.

Os campos importantes estão representado na Figura 4.27.

Type (camada Ethernet)

Indica que a mensagem utiliza IPv4.

Protocol (camada IPv4)

Indica que a mensagem encapsula um pacote IPv6 (protocolo número 41 – 6in4).

Destination (camada IPv4)

O destino é o endereço IPv4 do n4Relay (203.0.113.1).

```

3 0.000009 2001:db8:cb00:7182::b1c0 2012:2::6:12 ICMPv6 138 Echo (ping) request id=0x0035, seq=1
  Frame 3: 138 bytes on wire (1104 bits), 138 bytes captured (1104 bits)
  Ethernet II, Src: 00:00:00_aa:00:02 (00:00:00:aa:00:02), Dst: 00:00:00_aa:00:03 (00:00:00:aa:00:03)
    Destination: 00:00:00_aa:00:03 (00:00:00:aa:00:03)
    Source: 00:00:00_aa:00:02 (00:00:00:aa:00:02)
    Type: IP (0x0800)
  Internet Protocol Version 4, Src: 203.0.113.130 (203.0.113.130), Dst: 203.0.113.1 (203.0.113.1)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 124
    Identification: 0x0000 (0)
    Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: IPv6 (41)
    Header checksum: 0xc1d4 [correct]
    Source: 203.0.113.130 (203.0.113.130)
    Destination: 203.0.113.1 (203.0.113.1)
  Internet Protocol Version 6, Src: 2001:db8:cb00:7182::b1c0 (2001:db8:cb00:7182::b1c0), Dst: 2012:2::6:12 (2012:2::6:12)
    0110 .... = Version: 6
    .... 0000 0000 .... = Traffic class: 0x00000000
    .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 64
    Next header: ICMPv6 (0x3a)
    Hop limit: 63
    Source: 2001:db8:cb00:7182::b1c0 (2001:db8:cb00:7182::b1c0)
    Destination: 2012:2::6:12 (2012:2::6:12)
  Internet Control Message Protocol v6
  
```

```

0020  71 01 60 00 00 00 00 40 3a 3f 20 01 0d b8 cb 00  q:....@ :? .....
0030  71 82 00 00 00 00 00 b1 c0 20 12 00 02 00 00  4:.....
0040  00 00 00 00 00 00 00 06 00 12 80 00 81 bf 00 35  .....5
0050  00 01 07 53 b4 52 0d be 0d 00 08 09 0a 0b 0c 0d  ..S.R.....
  
```

Figura 4.27: conteúdo de um pacote echo request / echo reply, com conteúdo IPv6 encapsulado em IPv4 (/64).

Source (camada IPv4)

A origem é o endereço IPv4 do n2CPE (203.0.113.130).

Destination (camada IPv6)

O destino é o endereço IPv6 de n5Host (2012:2::6:12).

Source (camada IPv6)

A origem é o endereço IPv6 do n1Client (2001:db8:cb00:7182::b1c0).

9. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 4.5. IPv6 Rapid Deployment (6rd): configuração 6rd em um CPE (/56)

Objetivo

Esta experiência demonstra o funcionamento da técnica *IPv6 Rapid Deployment* (6rd), que permite oferecer conectividade IPv6 aos usuários através de um túnel IPv4. Para tanto, será efetuada a configuração manual do *relay* na rede do provedor e de CPEs para três diferentes usuários, fornecendo uma rede /56 para cada um.

Será utilizada a topologia descrita no arquivo: **4-05-6rd-multiple-CPEs.imn**.

Introdução teórica

Ver introdução teórica da Experiência 4.4.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **4-05-6rd-multiple-CPEs.imn** localizado no diretório lab, dentro do Desktop. A topologia da rede, representada pela Figura 4.28, deve aparecer.

Nesta topologia há três conjuntos de CPEs e clientes. Os CPEs n6CPE2 e n8CPE3 já estão configurados, bem como o n2Relay.

O cliente receberá um prefixo /56, obtido a partir de um prefixo IPv6 /48 do provedor e do endereço IPv4.

Isso é possível porque o prefixo de rede IPv4 é omitido na formação do prefixo IPv6 atribuído ao cliente no 6rd, como explicado na introdução teórica.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 e IPv4 nos nós n1Host, n2Relay, n3Router, n4CPE1, n5ClientA, n6CPE2, n7ClientB, n8CPE3 e n9ClientC.
 - (a) Verifique que o roteador n3Router não suporta IPv6. Observa-se que não há endereços IPv6 nas interfaces, nem mesmo endereços do tipo *link-local*, o que indica que o roteador não suporta o protocolo.

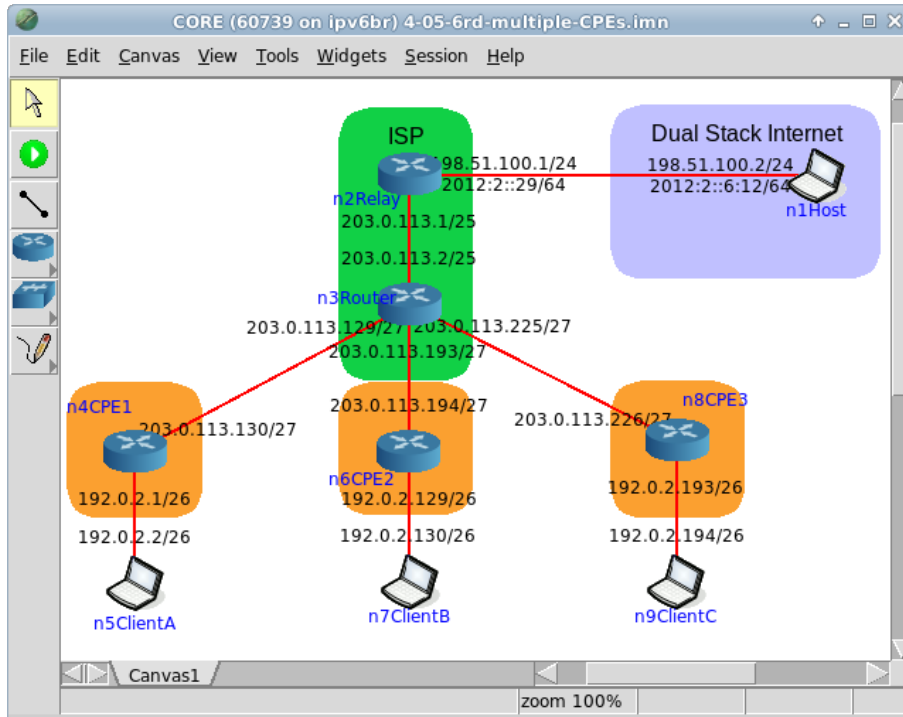


Figura 4.28: topologia da Experiência 4.5 no CORE.

- (b) Ao verificar a configuração de endereços em n7ClientB, pode-se observar que o endereço 2001:db8:cafe:c200::abcd/64 está configurado na interface que se comunica com o n6CPE2, que recebeu a rede 2001:db8:cafe:c200::/56.
- (c) Ao verificar a configuração de endereços em n9ClientC, pode-se observar que o endereço 2001:db8:cafe:e200::1234/64 está configurado na interface que se comunica com o n8CPE, que recebeu a rede 2001:db8:cafe:e200::/56.
- (d) Ao verificar a configuração de endereços em n4CPE1, pode-se observar que, no contexto do IPv6, há somente os endereços *link-local* e o *loopback* configurados.

3. Verifique a conectividade entre os clientes.

- (a) Utilize os seguintes comandos em `n7ClientB` para verificar a conectividade com `n9ClientC` e `n1Host`:

```
# ping -c 2 192.0.2.194
# ping6 -c 2 2001:db8:cafe:e200::1234
# ping -c 2 198.51.100.2
# ping6 -c 2 2012:2::6:12
# ip -6 route show
```

O resultado do comando é representado na Figura 4.29.

Observe que há conectividade IPv4 e IPv6 partindo de `n7ClientB`, tanto internamente quanto externamente em relação à rede do ISP. Pode-se verificar por meio do último comando que existe uma rota padrão utilizada para pacotes IPv6.

- (b) Utilize os seguintes comandos em `n5ClientA` para verificar a conectividade com `n7ClientB` e `n1Host`:

```
# ping -c 2 192.0.2.130
# ping6 -c 2 2001:db8:cafe:c200::abcd
# ping -c 2 198.51.100.2
# ping6 -c 2 2012:2::6:12
# ip -6 route show
```

O resultado do comando é representado na Figura 4.30.

Note que há conectividade IPv4, mas não há ainda conectividade IPv6. A configuração de `6rd` proverá essa conectividade. A falta de conectividade pode ser confirmada por meio do último comando, que indica existência de rota somente para endereços do tipo *link-local*.

```

n7ClientB
root@n7ClientB:/tmp/pycore.40450/n7ClientB.conf# ping -c 2 192.0.2.194
PING 192.0.2.194 (192.0.2.194) 56(84) bytes of data.
64 bytes from 192.0.2.194: icmp_req=1 ttl=61 time=0.142 ms
64 bytes from 192.0.2.194: icmp_req=2 ttl=61 time=0.098 ms

--- 192.0.2.194 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.098/0.120/0.142/0.022 ms
root@n7ClientB:/tmp/pycore.40450/n7ClientB.conf# ping6 -c 2 2001:db8:cafe:e200::1234
PING 2001:db8:cafe:e200::1234(2001:db8:cafe:e200::1234) 56 data bytes
64 bytes from 2001:db8:cafe:e200::1234: icmp_seq=1 ttl=62 time=0.272 ms
64 bytes from 2001:db8:cafe:e200::1234: icmp_seq=2 ttl=62 time=0.150 ms

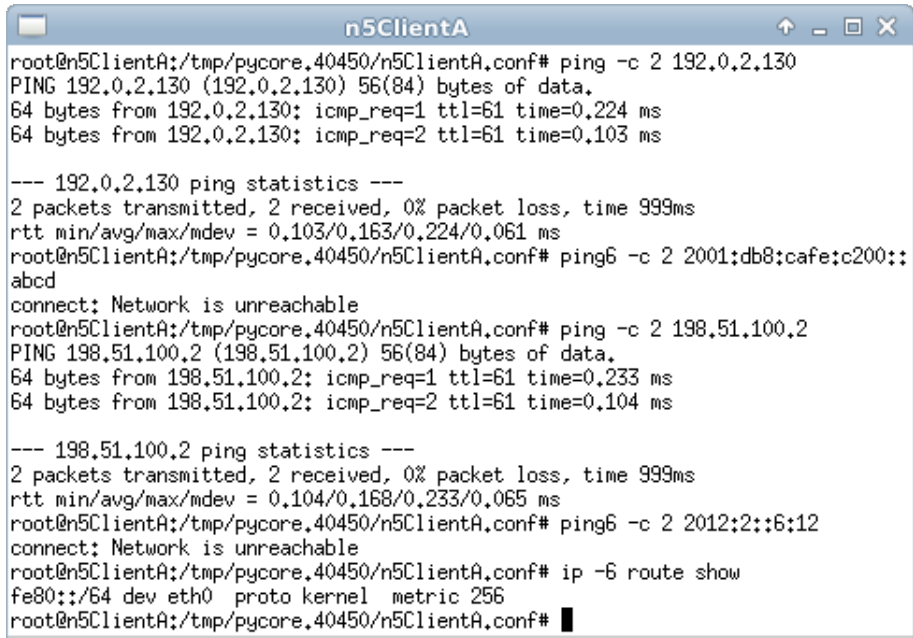
--- 2001:db8:cafe:e200::1234 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.150/0.211/0.272/0.061 ms
root@n7ClientB:/tmp/pycore.40450/n7ClientB.conf# ping -c 2 198.51.100.2
PING 198.51.100.2 (198.51.100.2) 56(84) bytes of data.
64 bytes from 198.51.100.2: icmp_req=1 ttl=61 time=0.306 ms
64 bytes from 198.51.100.2: icmp_req=2 ttl=61 time=0.103 ms

--- 198.51.100.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.103/0.204/0.306/0.102 ms
root@n7ClientB:/tmp/pycore.40450/n7ClientB.conf# ping6 -c 2 2012:2::6:12
PING 2012:2::6:12(2012:2::6:12) 56 data bytes
64 bytes from 2012:2::6:12: icmp_seq=1 ttl=62 time=0.171 ms
64 bytes from 2012:2::6:12: icmp_seq=2 ttl=62 time=0.134 ms

--- 2012:2::6:12 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.134/0.152/0.171/0.022 ms
root@n7ClientB:/tmp/pycore.40450/n7ClientB.conf# ip -6 route show
2001:db8:cafe:c200::/64 dev eth0 proto kernel metric 256
fe80::/64 dev eth0 proto kernel metric 256
default via 2001:db8:cafe:c200::1 dev eth0 metric 1024
root@n7ClientB:/tmp/pycore.40450/n7ClientB.conf# █

```

Figura 4.29: teste de conectividade do n7ClientB do brd (/56) e outros nós.



```

root@n5ClientA:/tmp/pycore.40450/n5ClientA.conf# ping -c 2 192.0.2.130
PING 192.0.2.130 (192.0.2.130) 56(84) bytes of data.
64 bytes from 192.0.2.130: icmp_req=1 ttl=61 time=0.224 ms
64 bytes from 192.0.2.130: icmp_req=2 ttl=61 time=0.103 ms

--- 192.0.2.130 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.103/0.163/0.224/0.061 ms
root@n5ClientA:/tmp/pycore.40450/n5ClientA.conf# ping6 -c 2 2001:db8:cafe:c200::abcd
connect: Network is unreachable
root@n5ClientA:/tmp/pycore.40450/n5ClientA.conf# ping -c 2 198.51.100.2
PING 198.51.100.2 (198.51.100.2) 56(84) bytes of data.
64 bytes from 198.51.100.2: icmp_req=1 ttl=61 time=0.233 ms
64 bytes from 198.51.100.2: icmp_req=2 ttl=61 time=0.104 ms

--- 198.51.100.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.104/0.168/0.233/0.065 ms
root@n5ClientA:/tmp/pycore.40450/n5ClientA.conf# ping6 -c 2 2012:2::6:12
connect: Network is unreachable
root@n5ClientA:/tmp/pycore.40450/n5ClientA.conf# ip -6 route show
fe80::/64 dev eth0 proto kernel metric 256
root@n5ClientA:/tmp/pycore.40450/n5ClientA.conf# █

```

Figura 4.30: teste de conectividade do n5ClientA do 6rd (/56) e outros nós.

4. Configure o 6rd no n4CPE1 e um endereço IPv6 no n5ClientA.
 - (a) Neste experimento, o *relay* já está configurado. Os seguintes comandos encontram-se nesse roteiro a título de comparação da configuração utilizada em relação ao experimento anterior.

```

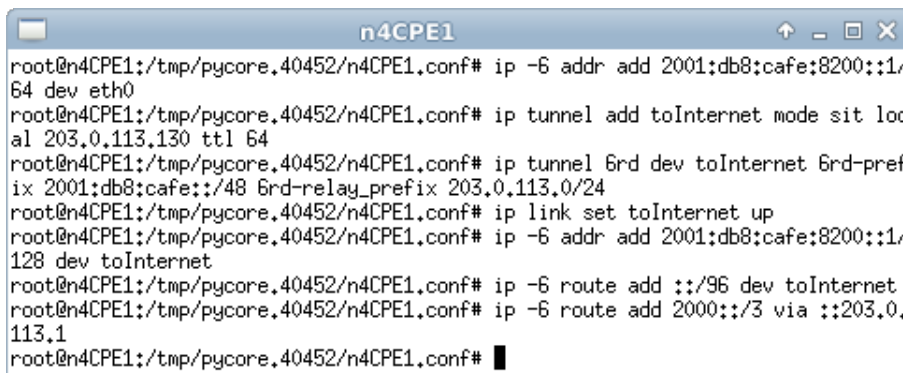
# ip tunnel add toClient mode sit local 203.0.113.1 ttl 64
# ip tunnel 6rd dev toClient 6rd-prefix
    2001:db8:cafe::/48 6rd-relay_prefix 203.0.113.0/24
# ip link set toClient up
# ip -6 addr add 2001:db8:cafe::1/128 dev toClient
# ip -6 route add 2001:db8:cafe::/48 dev toClient
# ip -6 route add 2000::/3 dev eth1

```

- (b) Abra o terminal de n4CPE1 com um duplo-clique e utilize os seguintes comandos para a configuração:

```
# ip -6 addr add 2001:db8:cafe:8200::1/64 dev eth0
# ip tunnel add toInternet mode sit local 203.0.113.130 ttl 64
# ip tunnel 6rd dev toInternet 6rd-prefix
  2001:db8:cafe::/48 6rd-relay_prefix 203.0.113.0/24
# ip link set toInternet up
# ip -6 addr add 2001:db8:cafe:8200::1/128
  dev toInternet
# ip -6 route add ::/96 dev toInternet
# ip -6 route add 2000::/3 via ::203.0.113.1
```

O resultado do comando é representado na Figura 4.31.



```
n4CPE1
root@n4CPE1:/tmp/pycore.40452/n4CPE1.conf# ip -6 addr add 2001:db8:cafe:8200::1/
64 dev eth0
root@n4CPE1:/tmp/pycore.40452/n4CPE1.conf# ip tunnel add toInternet mode sit loc
al 203.0.113.130 ttl 64
root@n4CPE1:/tmp/pycore.40452/n4CPE1.conf# ip tunnel 6rd dev toInternet 6rd-pref
ix 2001:db8:cafe::/48 6rd-relay_prefix 203.0.113.0/24
root@n4CPE1:/tmp/pycore.40452/n4CPE1.conf# ip link set toInternet up
root@n4CPE1:/tmp/pycore.40452/n4CPE1.conf# ip -6 addr add 2001:db8:cafe:8200::1/
128 dev toInternet
root@n4CPE1:/tmp/pycore.40452/n4CPE1.conf# ip -6 route add ::/96 dev toInternet
root@n4CPE1:/tmp/pycore.40452/n4CPE1.conf# ip -6 route add 2000::/3 via ::203.0.
113.1
root@n4CPE1:/tmp/pycore.40452/n4CPE1.conf# █
```

Figura 4.31: comandos de configuração do n4CPE1 do 6rd (/56).

Comparando a configuração deste exercício com a do experimento anterior, nota-se que o neste exercício o prefixo IPv6 do n4CPE1 é composto pelo prefixo IPv6 divulgado pelo indexISP ISP, o 2001:db8:cafe::/48, mais o final do endereço IPv4 do n4CPE1, descartando a porção referente à rede interna do ISP. Ou seja, do endereço 203.0.113.130/24 extrai-se o campo 130, em decimal, o converte para 0x82, em hexadecimal, resultando no prefixo IPv6 2001:db8:cafe:8200::/56.

- (c) Abra o terminal de `n5ClientA`, com um duplo-clique, e utilize os seguintes comandos para a configuração:

```
# ip -6 addr add 2001:db8:cafe:8200::abc:123/64 dev eth0
# ip -6 route add default via 2001:db8:cafe:8200::1
```

O resultado do comando é representado na Figura 4.32.



```
n5ClientA
root@n5ClientA:/tmp/pycore.40452/n5ClientA.conf# ip -6 addr add 2001:db8:cafe:8200::abc:123/64 dev eth0
root@n5ClientA:/tmp/pycore.40452/n5ClientA.conf# ip -6 route add default via 2001:db8:cafe:8200::1
root@n5ClientA:/tmp/pycore.40452/n5ClientA.conf# █
```

Figura 4.32: comandos de configuração do `n5ClientA` do `6rd` (/56).

Neste experimento, os valores apresentados na introdução teórica de `n`, `o` e `m` são, respectivamente, 48, 8 e 8 *bits*. Nesta configuração, é atribuído ao cliente um prefixo IPv6 /56, permitindo a criação de 256 sub-redes /64 ($2^8 = 256$). Destas, foi escolhida para a rede entre `n4CPE1` e `n5ClientA` foi a primeira, a `2001:db8:cafe:8200::/64`.

5. Verifique a conectividade IPv6 partindo de `n5ClientA`.

- (a) Abra o terminal de `n5ClientA` e utilize os seguintes comandos para verificar a conectividade com `n7ClientB`, `n9ClientC` e `n1Host`:

```
# ping6 -c 2 2001:db8:cafe:c200::abcd
# ping6 -c 2 2001:db8:cafe:e200::1234
# ping6 -c 2 2012:2::6:12
# traceroute6 2001:db8:cafe:e200::1234
# traceroute6 2012:2::6:12
```

O resultado do comando é representado na Figura 4.33.

```

n5ClientA
root@n5ClientA:/tmp/pycore.40452/n5ClientA.conf# ping6 -c 2 2001:db8:cafe:c200::
abcd
PING 2001:db8:cafe:c200::abcd(2001:db8:cafe:c200::abcd) 56 data bytes
64 bytes from 2001:db8:cafe:c200::abcd: icmp_seq=1 ttl=62 time=0,355 ms
64 bytes from 2001:db8:cafe:c200::abcd: icmp_seq=2 ttl=62 time=0,139 ms

--- 2001:db8:cafe:c200::abcd ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0,139/0,247/0,355/0,108 ms
root@n5ClientA:/tmp/pycore.40452/n5ClientA.conf# ping6 -c 2 2001:db8:cafe:e200::
1234
PING 2001:db8:cafe:e200::1234(2001:db8:cafe:e200::1234) 56 data bytes
64 bytes from 2001:db8:cafe:e200::1234: icmp_seq=1 ttl=62 time=0,300 ms
64 bytes from 2001:db8:cafe:e200::1234: icmp_seq=2 ttl=62 time=0,124 ms

--- 2001:db8:cafe:e200::1234 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0,124/0,212/0,300/0,088 ms
root@n5ClientA:/tmp/pycore.40452/n5ClientA.conf# ping6 -c 2 2012:2::6:12
PING 2012:2::6:12(2012:2::6:12) 56 data bytes
64 bytes from 2012:2::6:12: icmp_seq=1 ttl=62 time=0,227 ms
64 bytes from 2012:2::6:12: icmp_seq=2 ttl=62 time=0,159 ms

--- 2012:2::6:12 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0,159/0,193/0,227/0,034 ms
root@n5ClientA:/tmp/pycore.40452/n5ClientA.conf# traceroute6 2001:db8:cafe:e200:
:1234
traceroute to 2001:db8:cafe:e200::1234 (2001:db8:cafe:e200::1234) from 2001:db8:
cafe:8200::abc:123, port 33434, from port 65499, 30 hops max, 60 bytes packets
 1 2001:db8:cafe:8200::1 (2001:db8:cafe:8200::1) 0,005 ms 0,092 ms 0,120 ms
 2 2001:db8:cafe:e200::1 (2001:db8:cafe:e200::1) 0,364 ms 0,434 ms *
 3 2001:db8:cafe:e200::1234 (2001:db8:cafe:e200::1234) 0,725 ms * 0,003 ms
root@n5ClientA:/tmp/pycore.40452/n5ClientA.conf# traceroute6 2012:2::6:12
traceroute to 2012:2::6:12 (2012:2::6:12) from 2001:db8:cafe:8200::abc:123, port
33434, from port 65498, 30 hops max, 60 bytes packets
 1 2001:db8:cafe:8200::1 (2001:db8:cafe:8200::1) 0,005 ms 0,098 ms 0,122 ms
 2 2001:db8:cafe::1 (2001:db8:cafe::1) 0,336 ms 0,765 ms *
 3 2012:2::6:12 (2012:2::6:12) 1,084 ms * 0,004 ms
root@n5ClientA:/tmp/pycore.40452/n5ClientA.conf# █

```

Figura 4.33: teste de conectividade do n5ClientA.

6. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 4.6. NAT64: implantação utilizando TAYGA

Objetivo

Este laboratório apresenta a implantação do NAT64, uma técnica de transição que permite *hosts* somente IPv6 acessarem *hosts* somente IPv4 na Internet, por meio da tradução dos protocolos IP. O cenário inicial do laboratório apresentará a simplificação da rede de um ISP, configurada apenas com IPv6 na rede dos clientes e com o IPv4 na interface externa do roteador de borda e do servidor DNS recursivo. A tarefa do aluno será configurar:

1. O TAYGA, um *software* que realiza a tradução IPv6 para IPv4 de forma 1:1, traduzindo cada endereço IPv6 para um endereço IPv4 privado, no momento do envio de pacotes para um *host* IPv4 na Internet.
2. O iptables, que criará um NAT44 para associar todos os endereços IPv4 privados ao mesmo IPv4 público do roteador NAT64.
3. O BIND9, que funcionará como servidor DNS recursivo da rede e como servidor DNS64, convertendo respostas com registros do tipo A em registros do tipo AAAA.

Para o presente exercício, será utilizada a topologia descrita no arquivo:

4-06-NAT64.imn.

Introdução teórica

O NAT64 é uma técnica de transição que permite que nós somente IPv6 acessem nós IPv4 na Internet. Nesta técnica, uma rede IPv6 enxerga os endereços IPv4 da Internet como um endereço IPv6 construído a partir do prefixo do NAT64 e do endereço IPv4 de destino. Já os endereços IPv6 são vistos na Internet como um único IP: o endereço IPv4 do roteador de borda, que realiza um NAT de IPv6 para IPv4. Assim, já que para a Internet todos os nós IPv6 atrás do NAT64 estão representados pelo mesmo endereço IPv4, esta tradução é N:1. É também *stateful*, pois cria

estados para associar os endereços IPv6 ao IPv4 do NAT64. Esses estados servem para encaminhar os pacotes da Internet para o nó correto no momento do retorno destes pacotes da Internet. A Figura 4.34 demonstra a construção de um IPv6 traduzido do IPv4.

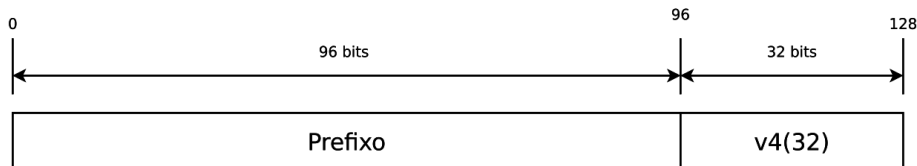


Figura 4.34: *endereçamento IPv6 traduzido do IPv4.*

O TAYGA é um *software open source* mantido nos repositórios de diversas distribuições Linux. Ele não trabalha com módulos do *kernel* do Linux, e sim com uma interface virtual TUN. Os pacotes IPv6 e IPv4 são encaminhados para esta interface, que então realiza a tradução dos mesmos e os reencaminha para a respectiva rede de destino. A interface TUN funciona como um roteador que intermedia a comunicação entre as redes IPv6 e IPv4 e por isto possui IPs em ambos os mundos.

Diferentemente da configuração original do NAT64, descrita anteriormente, o TAYGA não realiza uma tradução IPv6 para IPv4 de forma N:1, e sim 1:1. Cada nó IPv6 que envia um pacote para um *host* IPv4 por meio do TAYGA tem o seu endereço IPv6 associado a um endereço IPv4 diferente e **privado**. Por isso, o TAYGA pode ser considerado uma implementação incompleta do NAT64. Para implantar o mecanismo NAT64 de maneira completa com o TAYGA, é necessário adicionar um NAT44 entre a saída da interface TUN e a interface externa do roteador. O NAT44 associará todos os endereços IPv4 privados, dados aos nós IPv6 pelo TAYGA, ao endereço IPv4 público do roteador de borda. A Figura 4.35 demonstra esta tradução IPv6 para IPv4 em duas etapas.

O NAT64 necessita de uma técnica auxiliar para a conversão do DNS, chamada de DNS64 pela RFC 6147 (Bagnulo *et al.*, 2011). São sistemas distintos, mas que trabalham em conjunto para permitir a comunicação entre as redes IPv6 e IPv4. Basicamente, ele atua verificando as consultas DNS e, se não há uma resposta AAAA para o domínio solicitado, ele converte a resposta A recebida em uma resposta AAAA, convertendo os endereços usando a mesma regra e pre-

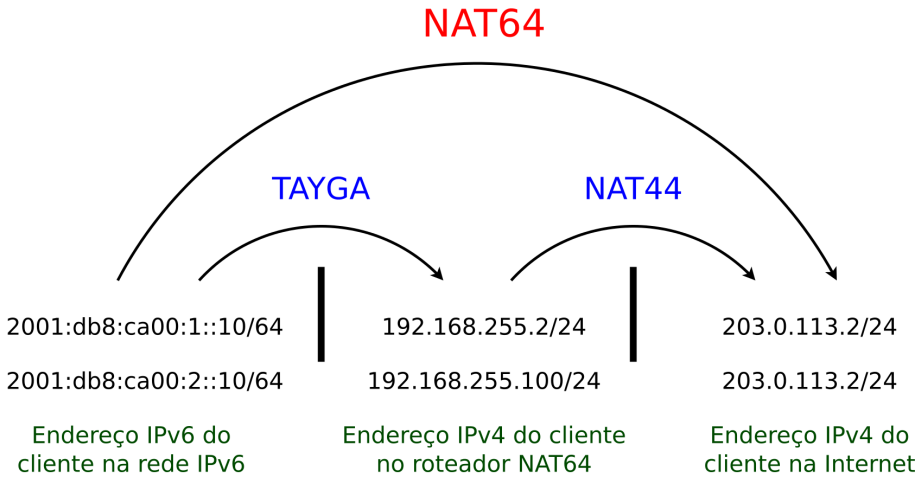


Figura 4.35: tradução IPv6 para IPv4 em duas etapas, realizada pelo NAT64 implantado com o TAYGA.

fixo do NAT64. As principais implementações do DNS64 são o BIND (<http://www.isc.org/software/bind>), que possui versões para Linux e Windows, e o Totd (<http://www.dillema.net/software/totd.html>), com versões para Linux e FreeBSD.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **4-06-NAT64.imn** localizado no diretório lab, dentro do Desktop. A topologia representada na Figura 4.36 será exibida.

Esta topologia ilustra a situação em que um determinado ISP possui uma rede com clientes unicamente IPv6 e deseja implantar a técnica NAT64 para que estes passem a ter acesso à Internet IPv4.

2. Conforme descrito nos Apêndices B e C, inicie a simulação, verifique a configuração de endereços IPv6 nos nós da rede do ISP e da rede IPv4 only e a conectividade entre eles.

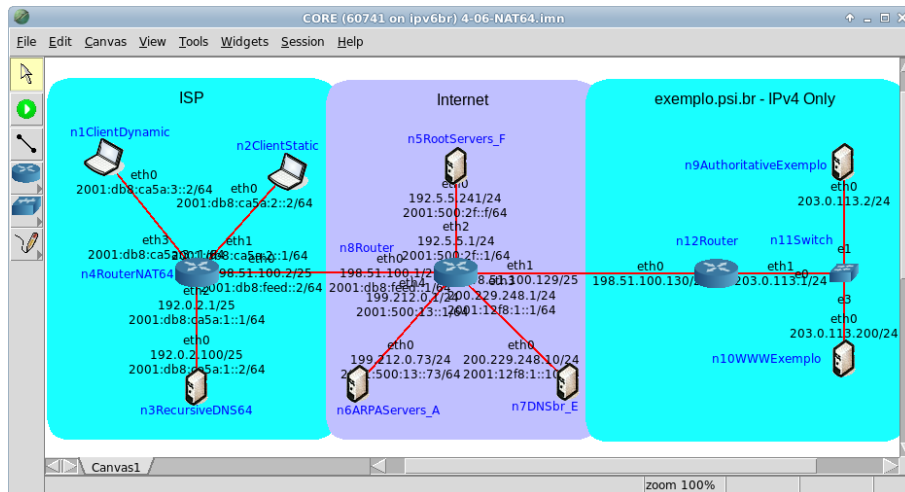


Figura 4.36: topologia da Experiência 4.6 no CORE.

Note que os clientes internos da rede do ISP possuem conectividade com o roteador de borda e que o roteador possui conectividade com a Internet, porém os clientes não são capazes de se comunicar nativamente com a rede IPv4 Only, pois os protocolos são incompatíveis. A comunicação se dará por meio do roteador de borda, por meio da tradução dos pacotes em ambos os sentidos.

3. Configure o TAYGA a fim de realizar a tradução IPv6 para IPv4.
 - (a) Abra um terminal de n4RouterNAT64 com um duplo-clique e crie o arquivo de configuração do TAYGA:

```
# touch tayga.conf
```

O resultado do comando é representado pela Figura 4.37.

```
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# touch tayga.conf
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf#
```

Figura 4.37: criação dos arquivos de configuração e inicialização do TAYGA.

- (b) Ainda no terminal de n4RouterNAT64, edite o arquivo recém-criado de configuração do TAYGA, o tayga.conf, de modo a inserir as seguintes linhas:

```
tun-device nat64
ipv4-addr 192.168.255.1
prefix 2001:db8:ca00::/96
dynamic-pool 192.168.255.0/24
data-dir /tmp
map 192.168.255.100 2001:db8:ca5a:2::2
```

Este arquivo possui seis parâmetros configuráveis, dois deles opcionais. A configuração que será utilizada neste exercício está apresentada a seguir, acompanhada da descrição de cada parâmetro:

`tun-device`

Define o nome da interface TUN que o TAYGA criará. O nome padrão `nat64` é recomendado, porém não é obrigatório.

`ipv4-addr`

Define o endereço IPv4 do TAYGA, atribuído à interface TUN descrita anteriormente. Este endereço deve estar dentro do *pool* de endereços IPv4 privados definido no parâmetro `dynamic-pool`, descrito mais adiante. Para este exercício, se usará o endereço 192.168.255.1.

`prefix`

Define o prefixo IPv6 que mapeará o endereço IPv4 do *host* que o nó IPv6 quer contatar. Este parâmetro deve ser um prefixo /96, que pode estar contido no bloco IPv6 do ISP ou utilizar o prefixo *bem conhecido* 64:ff9b::/96, reservado especificamente para a utilização em algoritmos de mapeamento de endereços IPv4 em IPv6. Por exemplo, o IPv4 203.0.113.200 seria convertido para o endereço IPv6 64:ff9b::203.0.113.200. Nesta experiência, será utilizado um prefixo da rede IPv6 do ISP, que consiste do bloco 2001:db8:ca00::/32, sendo escolhido o prefixo 2001:db8:ca00::/96.

`dynamic-pool`

Define o prefixo IPv4 de endereços privados para os quais o TAYGA traduz os endereços IPv6. Deve-se escolher um prefixo dentre os apresentados na RFC 1918 (Rekhter *et al.*, 1996). Foi escolhido para este exercício o prefixo 192.168.255.0/24, que faz parte o IPv4 do TAYGA.

data-dir

Este parâmetro não é obrigatório, porém é recomendável usá-lo, pois ele define o diretório que será utilizado para manter um arquivo chamado `dynamic.map`. Este arquivo guarda o estado das associações IPv6/IPv4-privado criadas dinamicamente pelo TAYGA. Assim, caso o sistema caia ou seja reiniciado e um pacote tenha sido enviado à Internet sem ainda ter retornado, o estado será recuperado após o sistema voltar, e assim o pacote poderá ser encaminhado corretamente ao nó IPv6 de origem quando retornar da Internet.

map

Este parâmetro também é opcional. Ele define um mapeamento estático entre um endereço IPv6 e um endereço IPv4 que é independente das regras de tradução estabelecidas nos outros parâmetros. Assim, se um pacote tiver como destino um destes dois endereços e for encaminhado à interface TUN do roteador, o TAYGA realizará automaticamente a tradução para o outro endereço e reencaminhará o pacote traduzido para o mundo IP correspondente (6 ou 4). Note, porém, que as regras de mapeamento estático são definidas individualmente, o que requererá uma linha com o parâmetro `map` para cada endereço IPv6/IPv4 que precisar de um mapeamento estático e/ou independente da regra de tradução configurada no TAYGA.

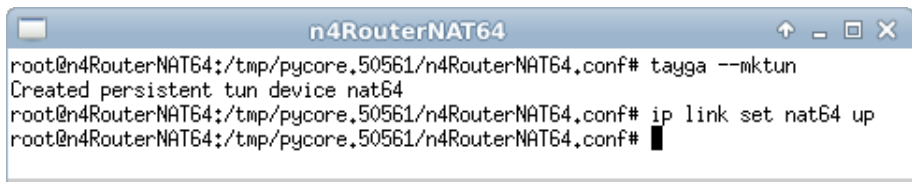
Neste exercício, será demonstrado por meio do segundo cliente do NAT64, `2001:db8:ca5a:2::2`, o efeito do mapeamento estático.

- (c) Ainda no terminal de n4RouterNAT64, são necessárias mais algumas configurações para o funcionamento correto do TAYGA. Para isto, execute os seguintes comandos:

- i. Crie uma interface TUN:

```
# tayga --mktun
# ip link set nat64 up
```

O resultado deverá ser como apresentado na Figura 4.38.



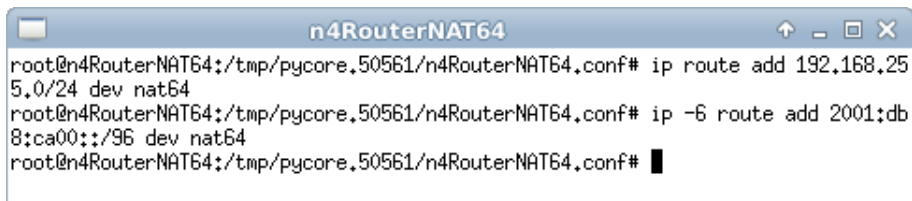
```
n4RouterNAT64
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# tayga --mktun
Created persistent tun device nat64
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# ip link set nat64 up
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# █
```

Figura 4.38: criação da interface TUN.

- ii. Configure o roteamento dos pacotes a serem traduzidos para a interface TUN:

```
# ip route add 192.168.255.0/24 dev nat64
# ip -6 route add 2001:db8:ca00::/96 dev nat64
```

O resultado deverá ser similar ao apresentado na Figura 4.39.



```
n4RouterNAT64
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# ip route add 192.168.255.0/24 dev nat64
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# ip -6 route add 2001:db8:ca00::/96 dev nat64
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# █
```

Figura 4.39: configuração das rotas do túnel.

- iii. Configure o iptables para a realização de um NAT44 entre os endereços IPv4 privados, roteados para a interface TUN nat64; e o endereço IPv4 público do roteador, roteado para a interface eth0:

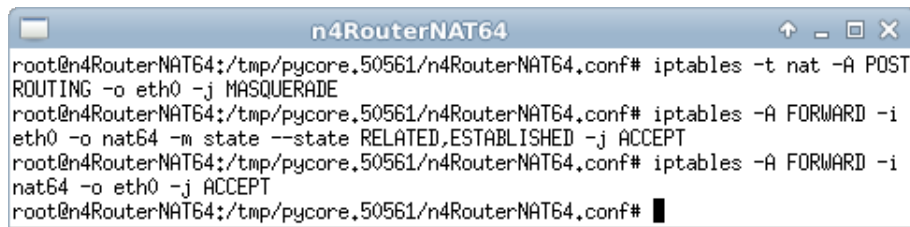
```
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
# iptables -A FORWARD -i eth0 -o nat64 -m state
  --state RELATED,ESTABLISHED -j ACCEPT
# iptables -A FORWARD -i nat64 -o eth0 -j ACCEPT
```

O resultado do comando é representado pela Figura 4.40.

- iv. Por fim, inicialize o TAYGA em modo de *debug* e utilize as configurações do arquivo `tayga.conf` descritas anteriormente:

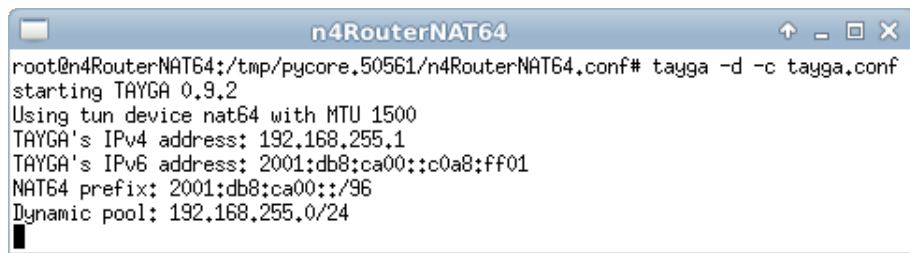
```
# tayga -d -c tayga.conf
```

O resultado do comando é representado pela Figura 4.41.



```
n4RouterNAT64
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# iptables -A FORWARD -i eth0 -o nat64 -m state --state RELATED,ESTABLISHED -j ACCEPT
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# iptables -A FORWARD -i nat64 -o eth0 -j ACCEPT
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# █
```

Figura 4.40: configuração do `iptables`.



```
n4RouterNAT64
root@n4RouterNAT64:/tmp/pycore.50561/n4RouterNAT64.conf# tayga -d -c tayga.conf
starting TAYGA 0.9.2
Using tun device nat64 with MTU 1500
TAYGA's IPv4 address: 192.168.255.1
TAYGA's IPv6 address: 2001:db8:ca00::c0a8:ff01
NAT64 prefix: 2001:db8:ca00::/96
Dynamic pool: 192.168.255.0/24
█
```

Figura 4.41: inicialização do TAYGA.

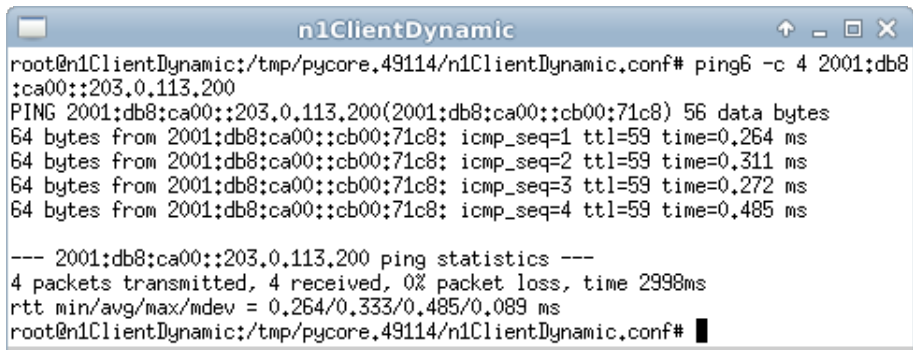
4. Conforme descrito no Apêndice C, realize o teste de conectividade entre o cliente IPv6 `n1ClientDynamic` e o *host* `n10WWWExemplo` da rede IPv4 Only.
5. Em paralelo, efetue:
 - (a) A coleta dos pacotes trafegados na interface `nat64` de `n4RouterNAT64`, utilizando o comando `tcpdump`. As instruções de coleta de pacotes utilizando `tcpdump` se encontram no Apêndice C.
 - (b) A verificação de conectividade IPv6 entre `n1ClientDynamic` e `n10WWWExemplo`. Para isto utilize o comando:

```
# ping6 -c 4 2001:db8:ca00::203.0.113.200
```

O resultado do comando é representado pela Figura 4.42.

- (c) Analise a saída do terminal do cliente `n1ClientDynamic`.

No terminal do cliente, pode-se ver o comando `ping6` em execução. Perceba que este comando permite escrever um endereço IPv6 traduzido do IPv4, usando diretamente o endereço IPv4 em formato decimal (203.0.113.200) e não em hexadecimal (cb00:71c8). Isto facilita o uso de comandos de rede no NAT64. Perceba também que as mensagens geradas pelo `ping6` convertem este endereço IPv4 em decimal para o formato correto, em hexadecimal. Finalmente, confirme neste terminal o retorno dos pacotes.



```

root@n1ClientDynamic:/tmp/pycore.49114/n1ClientDynamic.conf# ping6 -c 4 2001:db8
:ca00::203.0.113.200
PING 2001:db8:ca00::203.0.113.200(2001:db8:ca00::cb00:71c8) 56 data bytes
64 bytes from 2001:db8:ca00::cb00:71c8: icmp_seq=1 ttl=59 time=0.264 ms
64 bytes from 2001:db8:ca00::cb00:71c8: icmp_seq=2 ttl=59 time=0.311 ms
64 bytes from 2001:db8:ca00::cb00:71c8: icmp_seq=3 ttl=59 time=0.272 ms
64 bytes from 2001:db8:ca00::cb00:71c8: icmp_seq=4 ttl=59 time=0.485 ms

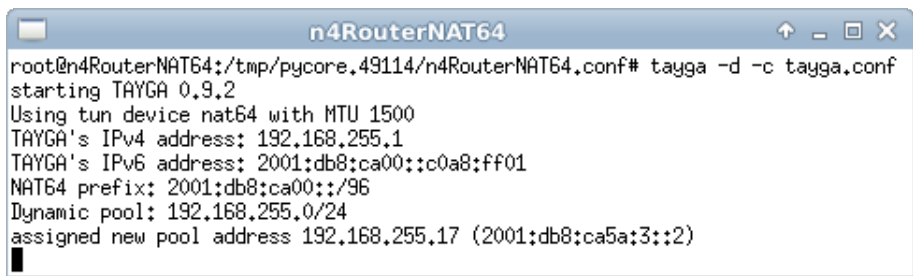
--- 2001:db8:ca00::203.0.113.200 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.264/0.333/0.485/0.089 ms
root@n1ClientDynamic:/tmp/pycore.49114/n1ClientDynamic.conf# █

```

Figura 4.42: *verificação de conectividade entre um cliente IPv6 da rede com NAT64 e um host IPv4 da Internet, com alocação dinâmica de endereço IPv4 privado.*

- (d) Analise a saída do terminal do roteador `n4RouterNAT64`.

No terminal do roteador, perceba a geração de uma nova linha de texto no final similar ao apresentado na Figura 4.43.



```

root@n4RouterNAT64:/tmp/pycore.49114/n4RouterNAT64.conf# tayga -d -c tayga.conf
starting TAYGA 0.9.2
Using tun device nat64 with MTU 1500
TAYGA's IPv4 address: 192.168.255.1
TAYGA's IPv6 address: 2001:db8:ca00::c0a8:ff01
NAT64 prefix: 2001:db8:ca00::/96
Dynamic pool: 192.168.255.0/24
assigned new pool address 192.168.255.17 (2001:db8:ca5a:3::2)
█

```

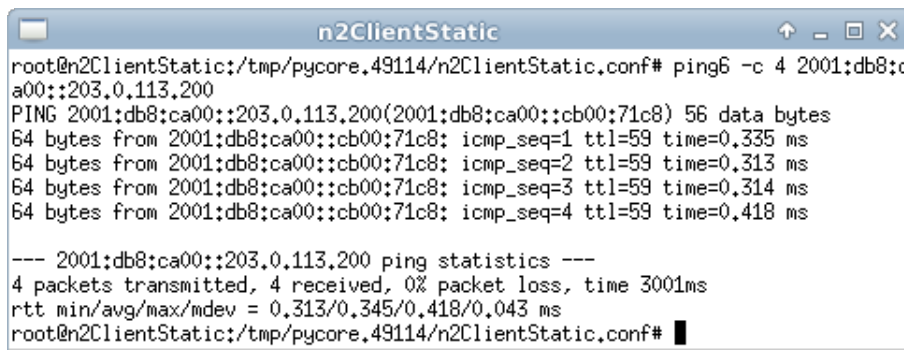
Figura 4.43: *alocação dinâmica de um endereço IPv4 privado a um cliente IPv6 da rede com NAT64.*

Esta linha demonstra a alocação dinâmica de um endereço IPv4 privado, retirado do *pool* definido anteriormente para o cliente IPv6. Esta alocação dinâmica é realizada por padrão pelo TAYGA, com o intuito de racionalizar o uso dos endereços IPv4 privados, que podem ser no futuro menor que a quantidade de clientes IPv6. A alocação permanece válida enquanto o cliente estiver usando o roteador para acessar a Internet IPv4 e dura por mais duas horas depois que o uso se encerra.

6. Realize o teste de conectividade entre o cliente IPv6 com mapeamento estático `n2ClientStatic` e o *host* `n10WWWExemplo`.
 - (a) Abra um terminal do cliente `n2ClientStatic` com um duplo clique e envie pacotes de `ping6` para o cliente `n10WWWExemplo` da rede IPv4 only, cujo endereço é `203.0.113.200`:

```
# ping6 -c 4 2001:db8:ca00::203.0.113.200
```

O resultado do comando é representado pela Figura 4.44.



```
n2ClientStatic
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# ping6 -c 4 2001:db8:ca00::203.0.113.200
PING 2001:db8:ca00::203.0.113.200(2001:db8:ca00::cb00:71c8) 56 data bytes
64 bytes from 2001:db8:ca00::cb00:71c8: icmp_seq=1 ttl=59 time=0.335 ms
64 bytes from 2001:db8:ca00::cb00:71c8: icmp_seq=2 ttl=59 time=0.313 ms
64 bytes from 2001:db8:ca00::cb00:71c8: icmp_seq=3 ttl=59 time=0.314 ms
64 bytes from 2001:db8:ca00::cb00:71c8: icmp_seq=4 ttl=59 time=0.418 ms

--- 2001:db8:ca00::203.0.113.200 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.313/0.345/0.418/0.043 ms
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf#
```

Figura 4.44: *verificação de conectividade entre um cliente IPv6 da rede com NAT64 e um host IPv4 da Internet, com mapeamento estático IPv6/IPv4.*

- (b) Analise a saída do terminal do cliente `n2ClientStatic`.
- (c) Analise a saída do terminal do roteador `n4RouterNAT64`.

No terminal do roteador `n4RouterNAT64`, perceba que não houve a geração da nova linha de texto que indica a alocação dinâmica de um endereço IPv4 privado para o cliente IPv6. Isto se deve ao fato de ter sido configurado um mapeamento estático do endereço IPv6 do cliente `n2ClientStatic` a um endereço IPv4 privado. Isso foi feito no arquivo `tayga.conf`, com o parâmetro `map`.

7. Efetue a análise dos pacotes coletados. A Figura 4.45 representa a captura de pacotes realizada.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	2001:db8:ca5a:3::2	2001:db8:ca00::cb00:7	ICMPv6	104	Echo (ping) request id=0x0059, seq=1
2	0.000059	192.168.255.17	203.0.113.200	ICMP	84	Echo (ping) request id=0x0059, seq=1/256, ttl=64
3	0.000173	203.0.113.200	192.168.255.17	ICMP	84	Echo (ping) reply id=0x0059, seq=1/256, ttl=64
4	0.000186	2001:db8:ca00::cb00:7	2001:db8:ca5a:3::2	ICMPv6	104	Echo (ping) reply id=0x0059, seq=1
5	0.999017	2001:db8:ca5a:3::2	2001:db8:ca00::cb00:7	ICMPv6	104	Echo (ping) request id=0x0059, seq=2
6	0.999136	192.168.255.17	203.0.113.200	ICMP	84	Echo (ping) request id=0x0059, seq=2/512, ttl=64
7	0.999227	203.0.113.200	192.168.255.17	ICMP	84	Echo (ping) reply id=0x0059, seq=2/512, ttl=64
8	0.999240	2001:db8:ca00::cb00:7	2001:db8:ca5a:3::2	ICMPv6	104	Echo (ping) reply id=0x0059, seq=2
9	1.999882	2001:db8:ca5a:3::2	2001:db8:ca00::cb00:7	ICMPv6	104	Echo (ping) request id=0x0059, seq=3
10	1.999934	192.168.255.17	203.0.113.200	ICMP	84	Echo (ping) request id=0x0059, seq=3/768, ttl=64
11	2.000041	203.0.113.200	192.168.255.17	ICMP	84	Echo (ping) reply id=0x0059, seq=3/768, ttl=64
12	2.000057	2001:db8:ca00::cb00:7	2001:db8:ca5a:3::2	ICMPv6	104	Echo (ping) reply id=0x0059, seq=3

Frame 1: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface
 Raw packet data
 Internet Protocol Version 6, Src: 2001:db8:ca5a:3::2 (2001:db8:ca5a:3::2), Dst: 2001:db8:ca00::cb00:71c8 (2001:db8:ca00::cb00:71c8)
 Internet Control Message Protocol v6

0000 60 00 00 00 00 40 3a 3f 20 01 0d b8 ca 5a 00 03@:~Z..
 0010 00 00 00 00 00 00 00 02 20 01 0d b8 ca 00 00 00
 0020 00 00 00 00 cb 00 71 c8 80 00 3d d4 00 59 00 01q:~..Y..
 0030 5f e0 84 53 3d 84 08 00 08 09 0a 0b 0c 0d 0e 0f ..S=.....

File: "/tmp/nat64-capture.pcap" 17... Packets: 16 Displayed: 16 Marked: 0 Load time: 0:00.000 Profile: Default

Figura 4.45: captura de pacotes realizada na interface `nat64` do roteador.

Perceba que a interface `nat64` reconhece duas requisições: uma de ping6 entre nós IPv6; e outra de ping entre nós IPv4. Uma análise dos endereços de origem e destino evidenciará que se trata da mesma requisição, que se origina no mundo IPv6, é traduzida para IPv4 e enviada à Internet, volta da Internet como IPv4 e então é traduzida de volta para IPv6 e enviada ao nó IPv6 de origem.

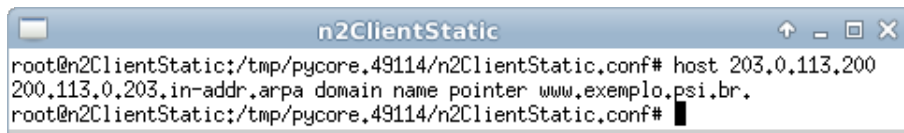
Pode-se ver no endereço de origem dos pacotes IPv4 do tipo *request* o endereço IPv4 privado que o nó IPv6 recebeu do TAYGA. Já no endereço de destino dos pacotes IPv6 do tipo *request* pode-se ver o endereço IPv6 traduzido do IPv4 do *host* na Internet. Nele se vê o prefixo do NAT64 e, no final do endereço, o IPv4 escrito em hexadecimal.

Até este ponto da configuração, é possível acessar *hosts* que só possuam endereços IPv4 utilizando apenas os endereços IPs literais. Para tornar a solução mais completa, configure o servidor DNS recursivo da rede do ISP para atuar como servidor DNS64. Com isto, será possível acessar *hosts* também por meio do nome de domínio, mesmo que este domínio possua apenas registros do tipo A.

8. Realize outro teste de conectividade entre o cliente `n2ClientStatic` e o *host* `n10WWWExemplo`, mas agora utilize o nome de domínio em lugar do endereço IP.
- (a) Para descobrir qual o nome de domínio do *host* `n10WWWExemplo`, abra um terminal do cliente `n2ClientStatic` e execute o seguinte comando:

```
# host 203.0.113.200
```

O resultado do comando é representado pela Figura 4.46.



The image shows a terminal window titled 'n2ClientStatic'. The prompt is 'root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf#'. The command entered is 'host 203.0.113.200'. The output is '200.113.0.203.in-addr.arpa domain name pointer www.exemplo.psi.br.'. The prompt is then followed by a cursor.

```
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# host 203.0.113.200
200.113.0.203.in-addr.arpa domain name pointer www.exemplo.psi.br.
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# █
```

Figura 4.46: verificação do nome de domínio do *host* `n10WWWExemplo`.

- (b) Ainda no terminal do cliente `n2ClientStatic`, envie pacotes de `ping6` para o *host* `n10WWWExemplo`. Utilize seu nome de domínio:

```
# ping6 -c 4 www.exemplo.psi.br
```

O resultado do comando é representado pela Figura 4.47.



```

root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# ping6 www.exemplo.psi.br
unknown host
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# █

```

Figura 4.47: teste de conectividade por meio do nome de domínio do host `n10WWWExemplo`.

Note que não foi possível estabelecer conexão com `n10WWWExemplo`, dado que não há endereços IPv6 associados a este domínio.

- (c) Para descobrir as informações disponíveis no serviço de DNS sobre o nome de domínio do *host* `n10WWWExemplo`, execute o seguinte comando no terminal do cliente `n2ClientStatic`:

```
# dig ANY www.exemplo.psi.br
```

O resultado do comando é representado pela Figura 4.48.

Note que não há nenhum registro AAAA associado ao domínio. Nem mesmo o *name server* do domínio é acessível por meio do IPv6.

9. Para resolver esta questão, configure o servidor DNS recursivo do ISP para que ele atue como servidor DNS64, traduzindo as respostas com registros do tipo A em respostas com registros do tipo AAAA. Nestas respostas, o endereço IPv4 obtido na consulta DNS será mapeado para o prefixo IPv6 configurado no NAT64, o `2001:db8:ca00::/96`, permitindo que a comunicação seja estabelecida.
- (a) Abra um terminal de `n3RecursiveDNS64` e edite o arquivo de configuração do BIND, localizado em `/etc/bind/named.conf`, de modo a acrescentar apenas o trecho destacado em **negrito**:

```

...
listen-on-v6 { any; };
dns64 2001:db8:ca00::/96 {
    clients { any; };
};
disable-empty-zone "2.0.192.in-addr.arpa";
...

```

```

n2ClientStatic
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# dig ANY www.exemplo.p
si.br

;<> DiG 9.8.1-P1 <> ANY www.exemplo.psi.br
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: NOERROR, id: 10896
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 2

;; QUESTION SECTION:
;www.exemplo.psi.br.          IN      ANY

;; ANSWER SECTION:
www.exemplo.psi.br.         172800 IN      MX      10 mail.exemplo.psi.br.
www.exemplo.psi.br.         172800 IN      A       203.0.113.200

;; AUTHORITY SECTION:
exemplo.psi.br.             170772 IN      NS      ns.exemplo.psi.br.

;; ADDITIONAL SECTION:
mail.exemplo.psi.br.        172800 IN      A       203.0.113.220
ns.exemplo.psi.br.          170772 IN      A       203.0.113.2

;; Query time: 6 msec
;; SERVER: 2001:db8:ca5a:1::2#53(2001:db8:ca5a:1::2)
;; WHEN: Tue May 27 16:16:34 2014
;; MSG SIZE rcvd: 122

root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# █

```

Figura 4.48: consulta DNS às informações sobre o domínio *www.exemplo.psi.br*.

- (b) Ainda no terminal de `n3RecursiveDNS64`, verifique se o arquivo de configuração foi gerado corretamente. Para isto execute o comando:

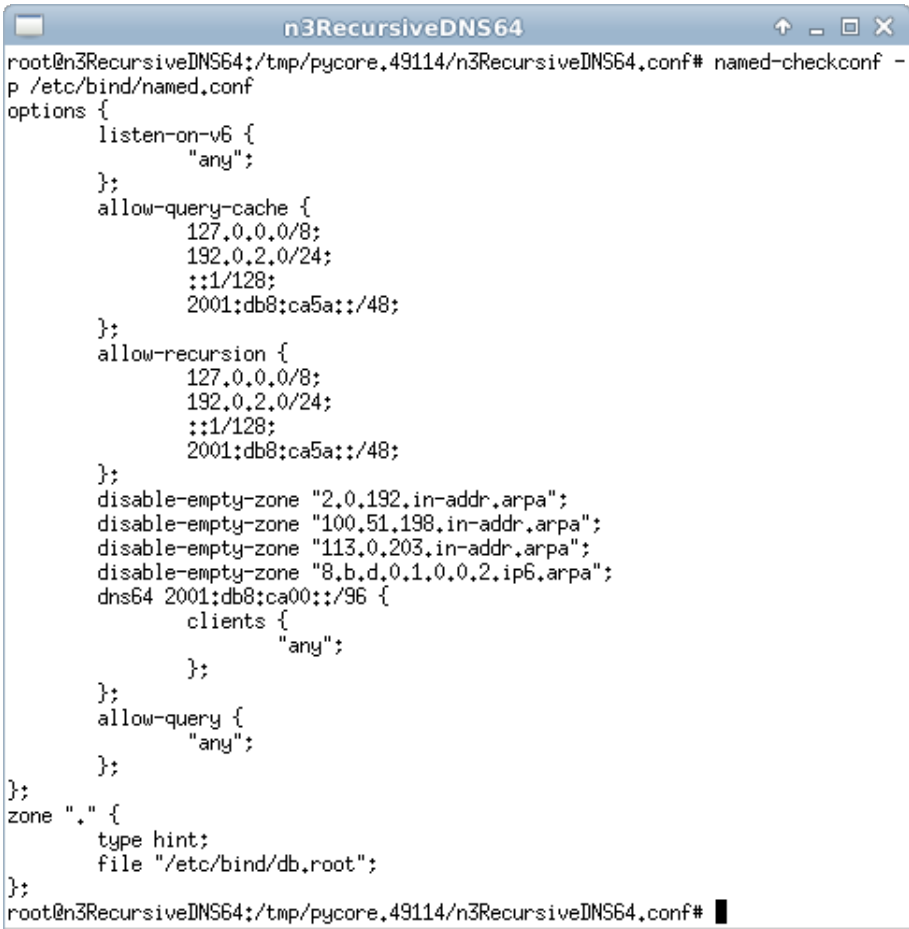
```
# named-checkconf -p /etc/bind/named.conf
```

O resultado do comando é representado pela Figura 4.49.

- (c) Caso o passo anterior não tenha apresentado erro de execução, reinicie o processo do BIND para que a alteração seja aplicada. Utilize os comandos:

```
# killall named
# named -c /etc/bind/named.conf
```

O resultado dos comandos é representado pela Figura 4.50.



```

root@n3RecursiveDNS64:/tmp/pycore.49114/n3RecursiveDNS64.conf# named-checkconf -
p /etc/bind/named.conf
options {
    listen-on-v6 {
        "any";
    };
    allow-query-cache {
        127.0.0.0/8;
        192.0.2.0/24;
        ::1/128;
        2001:db8:ca5a::/48;
    };
    allow-recursion {
        127.0.0.0/8;
        192.0.2.0/24;
        ::1/128;
        2001:db8:ca5a::/48;
    };
    disable-empty-zone "2.0.192.in-addr.arpa";
    disable-empty-zone "100.51.198.in-addr.arpa";
    disable-empty-zone "113.0.203.in-addr.arpa";
    disable-empty-zone "8.b.d.0.1.0.0.2.ip6.arpa";
    dns64 2001:db8:ca00::/96 {
        clients {
            "any";
        };
    };
    allow-query {
        "any";
    };
};
zone "." {
    type hint;
    file "/etc/bind/db.root";
};
root@n3RecursiveDNS64:/tmp/pycore.49114/n3RecursiveDNS64.conf# █

```

Figura 4.49: *verificação do arquivo de configuração do BIND após sua edição.*



```

root@n3RecursiveDNS64:/tmp/pycore.49114/n3RecursiveDNS64.conf# killall named
root@n3RecursiveDNS64:/tmp/pycore.49114/n3RecursiveDNS64.conf# named -c /etc/bin
d/named.conf
root@n3RecursiveDNS64:/tmp/pycore.49114/n3RecursiveDNS64.conf# █


```

Figura 4.50: *resultado da reinicialização do serviço DNS BIND.*

- (d) Efetue novamente uma consulta DNS para testar as novas configurações do servidor DNS recursivo da rede ISP. Abra um terminal de n2ClientStatic com um duplo-clique e efetue uma consulta por meio do comando:

```
# host www.exemplo.psi.br
```

O resultado do comando é representado pela Figura 4.51.



```

n2ClientStatic
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# host www.exemplo.psi.br
www.exemplo.psi.br has address 203.0.113.200
www.exemplo.psi.br has IPv6 address 2001:db8:ca00::cb00:71c8
www.exemplo.psi.br mail is handled by 10 mail.exemplo.psi.br.
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# █

```

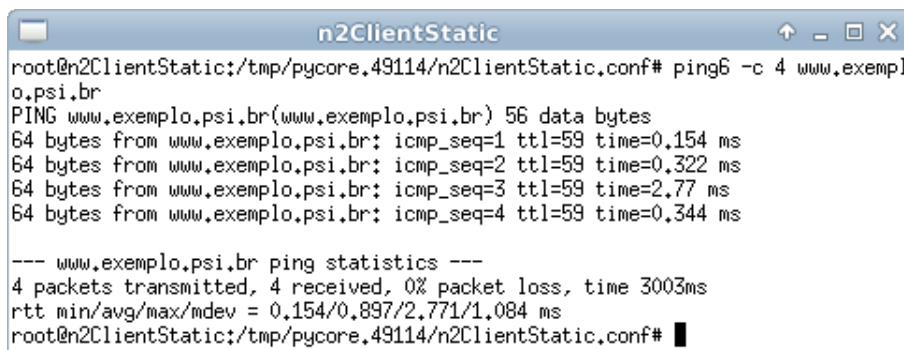
Figura 4.51: consulta DNS ao nome de domínio *www.exemplo.psi.br*.

Note que foi obtido como resposta à consulta DNS um endereço IPv6 associado ao domínio *www.exemplo.psi.br*. Este endereço foi traduzido pelo servidor DNS64, que recebeu originalmente como resposta um registro do tipo A, apontando o nome de domínio *www.exemplo.psi.br* ao endereço 203.0.113.200 e a traduziu para um registro do tipo AAAA, associando o domínio ao endereço IPv6 2001:db8:ca00::cb00:71c8. Perceba que os últimos 32 *bits* do endereço traduzido, o (cb00:71c8), representam o endereço IPv4 do *hosts* *n10WWWExemplo* convertido para hexadecimal.

- (e) Ainda no terminal do cliente *n2ClientStatic*, envie pacotes de ping6 para o *host* *n10WWWExemplo*. Utilize seu nome de domínio:

```
# ping6 -c 4 www.exemplo.psi.br
```

O resultado deve ser similar ao apresentado na Figura 4.52.



```

n2ClientStatic
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# ping6 -c 4 www.exemplo.psi.br
PING www.exemplo.psi.br(www.exemplo.psi.br) 56 data bytes
64 bytes from www.exemplo.psi.br: icmp_seq=1 ttl=59 time=0.154 ms
64 bytes from www.exemplo.psi.br: icmp_seq=2 ttl=59 time=0.322 ms
64 bytes from www.exemplo.psi.br: icmp_seq=3 ttl=59 time=2.77 ms
64 bytes from www.exemplo.psi.br: icmp_seq=4 ttl=59 time=0.344 ms

--- www.exemplo.psi.br ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.154/0.897/2.771/1.084 ms
root@n2ClientStatic:/tmp/pycore.49114/n2ClientStatic.conf# █

```

Figura 4.52: teste de conectividade por meio do nome de domínio do *host* *n10WWWExemplo*.

Também foi possível estabelecer a conexão entre `n2ClientStatic`, que só possui endereçamento IPv6 e `n10WWWExemplo`, que possui endereçamento apenas IPv4. Isto ocorreu devido à utilização das técnicas DNS64, que traduziram as respostas das consultas DNS e o NAT64, que traduziu os pacotes IPv6 em IPv4 e vice-versa.

Faça estes dois últimos testes também a partir do *host* `n1ClientDynamic` e veja se os resultados obtidos são os mesmos.

10. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 4.7. 464XLAT

Objetivo

Este laboratório apresenta o 464XLAT. Esta técnica de transição oferece uma forma de pilha dupla para clientes de uma rede somente IPv6, a fim de que acessem servidores somente IPv4 na Internet, por meio da tradução dos pacotes. O cenário inicial do laboratório apresentará uma expansão da topologia da experiência do NAT64. Haverá o acréscimo de um CPE (*Customer Premises Equipment*), que servirá como CLAT. O roteador NAT64, chamado de PLAT no 464XLAT, estará com a configuração final da experiência do NAT64. Desta forma, o 464XLAT será implantado como uma expansão natural do NAT64. A tarefa do aluno será configurar os seguintes itens:

1. O TAYGA, como um NAT46 *stateless*, no CLAT. Este elemento funcionará como CPE, fazendo a borda da LAN IPv4 do cliente.
2. As rotas de encaminhamento do CLAT; tanto as de encaminhamento de pacotes para tradução local, na interface TUN do TAYGA, quanto as de encaminhamento de pacotes para a tradução no PLAT.
3. Uma rota adicional no PLAT que direciona os pacotes de volta para o CLAT quando retornam da Internet.

Para o presente exercício será utilizada a topologia descrita no arquivo:
4-07-464XLAT.imn.

Introdução teórica

O 464XLAT é uma técnica de transição composta de duas outras técnicas: o NAT64 e o IVI. Assim como o NAT64, o objetivo do 464XLAT é permitir que nós com IPv6 nativo acessem *hosts* IPv4 na Internet. Porém, ao contrário da primeira técnica, os nós do 464XLAT possuem pilha dupla e utilizam o endereço real dos *hosts* IPv4 para enviar pacotes a eles.

Entre os clientes do 464XLAT e os *hosts* IPv4 na Internet existe uma rede somente IPv6. Por isso, é necessário realizar uma dupla tradução dos pacotes que um cliente do 464XLAT envia para um *host* IPv4, sendo de IPv4 para IPv6 e novamente para IPv4.

A tradução de IPv4 para IPv6 é realizada de forma *stateless* pelo CPE – o elemento equivalente ao IVI. A outra tradução, de IPv6 para IPv4, é realizada de forma *stateful* pelo roteador de borda da rede IPv6 do ISP – o elemento equivalente ao NAT64. No 464XLAT, o tradutor *stateless* recebe o nome de CLAT (*Customer Side Translator*), enquanto que o tradutor *stateful* recebe o nome de PLAT (*Provider Side Translator*). A Figura 4.53 apresenta um exemplo de topologia de rede do 464XLAT.

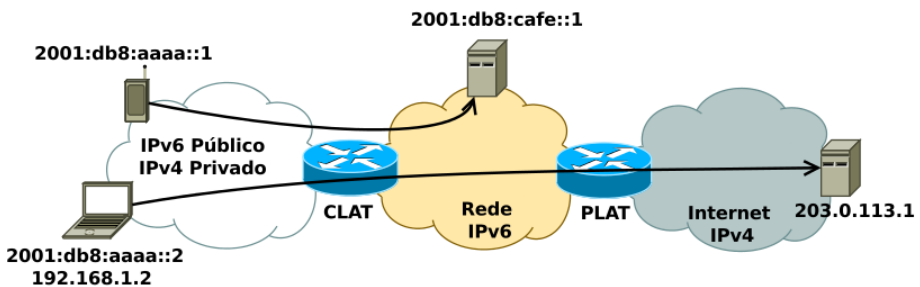


Figura 4.53: topologia do 464XLAT.

A pilha dupla dos clientes do 464XLAT consiste em um IPv6 público, com o qual se comunicam nativamente com *hosts* IPv6 na Internet; e um IPv4 privado, com o qual o cliente realiza uma comunicação em IPv4 com *hosts* da Internet. Porém, como já mencionado, não há roteamento IPv4 direto para a Internet, devido à rede somente IPv6 no caminho, o que torna a dupla tradução necessária, como pode ser observado na Figura 4.54.

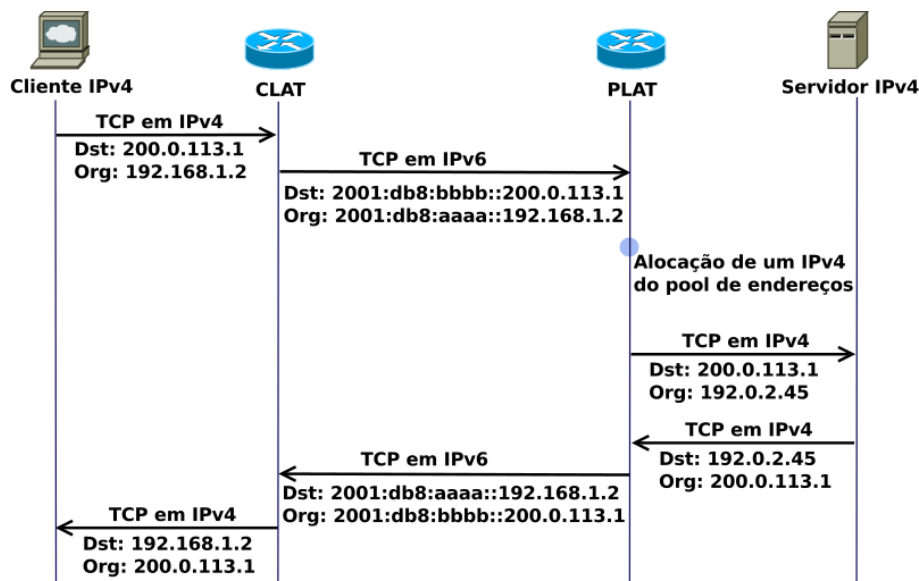


Figura 4.54: sequência de traduções do 464XLAT durante o envio e retorno de um pacote para um host IPv4 na Internet.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **4-07-464XLAT.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 4.55, deve aparecer.

Essa topologia ilustra a situação em que um determinado ISP possui uma rede com o NAT64 implantado e deseja expandi-lo para o 464XLAT. O PLAT já está configurado e continua utilizando o *software* TAYGA para realizar a tradução IPv6 para IPv4 de forma *stateful*, enquanto o CLAT utilizará o TAYGA para realizar a tradução IPv4 para IPv6 de forma *stateless*. Devido ao fato da rede IPv6 fazer fronteira com dois mundos IPv4 separados, o 464XLAT precisa de dois prefixos diferentes para os endereços IPv6 traduzidos do IPv4. O prefixo IPv6 associado aos endereços IPv4 dos serviços acessados na Internet será 2001:db8:ca00:bbbb::/96. Já o prefixo IPv6 associado aos endereços IPv4 dos clientes do 464XLAT será 2001:db8:ca00:aaaa::/96.

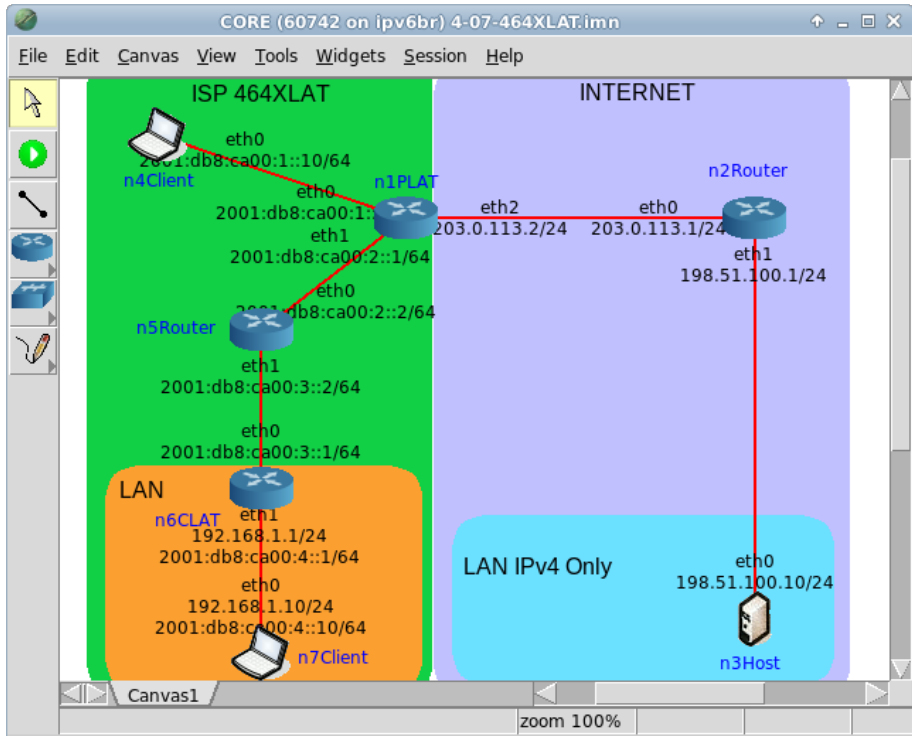


Figura 4.55: topologia da Experiência 4.7 no CORE.

O roteador n5Router já possui as rotas necessárias para os dois prefixos do 464XLAT: o prefixo dos IPv4 da Internet estará incluso na rota padrão do roteador que aponta para o PLAT e o prefixo do cliente do 464XLAT está roteado para o CLAT. Já o PLAT e o CLAT não terão inicialmente nenhuma das rotas relacionadas a estes prefixos, sendo necessário configurá-las junto com o TAYGA.

Para uma revisão da configuração do NAT64 pelo TAYGA, reveja a experiência do NAT64. A configuração do CLAT será apresentada a seguir.

- Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 nos nós n3Host, n4Client e n7Client e a conectividade entre eles.

3. Inicialize o TAYGA como um NAT64 no roteador PLAT a fim de realizar a tradução IPv6 para IPv4.
 - (a) Abra o terminal do roteador PLAT, com um duplo-clique sobre a máquina n1PLAT.
 - (b) Execute o *script* de inicialização do TAYGA:

```
# ./init-tayga64.sh start
```

O resultado dos comandos é representado pela Figura 4.56.



```
n1PLAT
root@n1PLAT:/tmp/pycore.38272/n1PLAT.conf# ./init-tayga64.sh start
Created persistent tun device nat64
starting TAYGA 0.9.2
Using tun device nat64 with MTU 1500
TAYGA's IPv4 address: 192.168.255.1
TAYGA's IPv6 address: 2001:db8:ca00:bbbb::c0a8:ff01
NAT64 prefix: 2001:db8:ca00:bbbb::/96
Dynamic pool: 192.168.255.0/24
Loaded 1 dynamic map from /tmp/dynamic.map
```

Figura 4.56: resultado da inicialização do TAYGA no PLAT, exibindo as configurações aplicadas.

Verifique as mensagens geradas pelo TAYGA em modo de *debug*.

- (c) Abra outro terminal do roteador n1PLAT e execute o comando a seguir, a fim de adicionar a rota para o prefixo do IPv6 traduzido do IPv4 do cliente do 464XLAT:

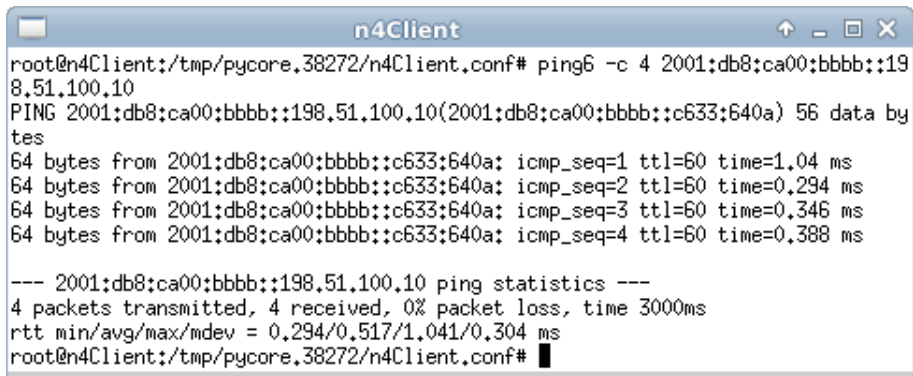
```
# ip -6 route add 2001:db8:ca00:aaaa::/96 via 2001:db8:ca00:2::2
```

Lembre-se de que o NAT64 não possui um nó equivalente ao CLAT em sua topologia e que o *script* utilizado na experiência do NAT64 não incluía a rota do prefixo do CLAT. Como o 464XLAT requer esta rota e o *script* executado anteriormente é baseado no *script* do NAT64, a criação da rota no PLAT foi feita manualmente.

4. Verifique a conectividade do cliente somente IPv6 do ISP com um *host* IPv4 da Internet.
 - (a) Abra o terminal do cliente somente IPv6 com um duplo-clique sobre a máquina *n4Client*.
 - (b) Com o comando `ping6` teste a conectividade IPv6 com a máquina *n3Host*. Utilize o prefixo de tradução IPv6 para IPv4:

```
# ping6 -c 4 2001:db8:ca00:bbbb::198.51.100.10
```

O resultado dos comandos é representado pela Figura 4.57.



```
root@n4Client:/tmp/pycore.38272/n4Client.conf# ping6 -c 4 2001:db8:ca00:bbbb::198.51.100.10
PING 2001:db8:ca00:bbbb::198.51.100.10(2001:db8:ca00:bbbb::c633:640a) 56 data bytes
64 bytes from 2001:db8:ca00:bbbb::c633:640a: icmp_seq=1 ttl=60 time=1.04 ms
64 bytes from 2001:db8:ca00:bbbb::c633:640a: icmp_seq=2 ttl=60 time=0.294 ms
64 bytes from 2001:db8:ca00:bbbb::c633:640a: icmp_seq=3 ttl=60 time=0.346 ms
64 bytes from 2001:db8:ca00:bbbb::c633:640a: icmp_seq=4 ttl=60 time=0.388 ms

--- 2001:db8:ca00:bbbb::198.51.100.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.294/0.517/1.041/0.304 ms
root@n4Client:/tmp/pycore.38272/n4Client.conf# █
```

Figura 4.57: *verificação de conectividade de um cliente somente IPv6 com um host IPv4 da Internet.*

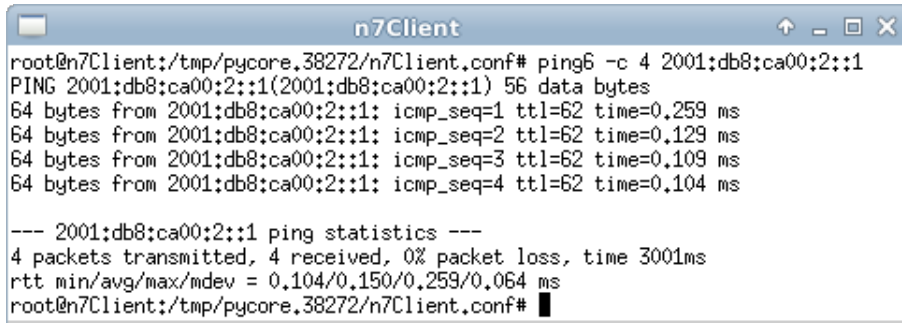
- (c) Feche o terminal do cliente *n4Client*.

Veja que o cliente interno da rede IPv6 possui conectividade com um *host* IPv4 por meio da tradução do PLAT. Para todos os efeitos, o PLAT continua sendo um simples NAT64 para os clientes somente IPv6 da rede do ISP.
5. Verifique a conectividade IPv6 e IPv4 do cliente com pilha dupla restrita do ISP.
 - (a) Abra o terminal do cliente do 464XLAT com um duplo-clique sobre a máquina *n7Client*.

- (b) Utilize o comando `ping6` para testar a conectividade IPv6 com a máquina `n1PLAT`:

```
# ping6 -c 4 2001:db8:ca00:2::1
```

O resultado dos comandos é representado pela Figura 4.58.



```
n7Client
root@n7Client:/tmp/pycore.38272/n7Client.conf# ping6 -c 4 2001:db8:ca00:2::1
PING 2001:db8:ca00:2::1(2001:db8:ca00:2::1) 56 data bytes
64 bytes from 2001:db8:ca00:2::1: icmp_seq=1 ttl=62 time=0,259 ms
64 bytes from 2001:db8:ca00:2::1: icmp_seq=2 ttl=62 time=0,129 ms
64 bytes from 2001:db8:ca00:2::1: icmp_seq=3 ttl=62 time=0,109 ms
64 bytes from 2001:db8:ca00:2::1: icmp_seq=4 ttl=62 time=0,104 ms

--- 2001:db8:ca00:2::1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0,104/0,150/0,259/0,064 ms
root@n7Client:/tmp/pycore.38272/n7Client.conf# █
```

Figura 4.58: *verificação de conectividade IPv6 de um cliente do 464XLAT.*

- (c) Utilize o comando `ping` para testar a conectividade IPv4 com a máquina `n3Host`:

```
# ping -c 4 198.51.100.10
```

O resultado do comando é representado pela Figura 4.59.



```
n7Client
root@n7Client:/tmp/pycore.38272/n7Client.conf# ping -c 4 198.51.100.10
PING 198.51.100.10 (198.51.100.10) 56(84) bytes of data.
From 192.168.1.1 icmp_seq=1 Destination Net Unreachable
From 192.168.1.1 icmp_seq=2 Destination Net Unreachable
From 192.168.1.1 icmp_seq=3 Destination Net Unreachable
From 192.168.1.1 icmp_seq=4 Destination Net Unreachable

--- 198.51.100.10 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 2999ms
root@n7Client:/tmp/pycore.38272/n7Client.conf# █
```

Figura 4.59: *verificação de conectividade IPv4 de um cliente do 464XLAT.*

Veja que o cliente do 464XLAT tem conectividade IPv6 normalmente, pois seu IPv6 é nativo, mas não é capaz de se comunicar com um *host* IPv4 na Internet, embora possua endereçamento IPv4 em sua interface. Isto se deve à ausência de uma rota direta em IPv4 para a Internet.

6. Configure o TAYGA como um NAT46 no CLAT a fim de realizar a tradução IPv4 para IPv6.
 - (a) Abra um terminal de n6CLAT com um duplo-clique e crie o arquivo de configuração do TAYGA por meio do comando:

```
# touch tayga46.conf
```

O resultado do comando é representado pela Figura 4.60.



Figura 4.60: *esultado da criação dos arquivos de configuração e inicialização do TAYGA.*

- (b) Ainda no terminal de n6CLAT, edite o arquivo de configuração do TAYGA, localizado em `tayga46.conf`, de modo a inserir as seguintes linhas:

```
tun-device nat64
ipv4-addr 192.168.2.1
prefix 2001:db8:ca00:bbbb::/96
dynamic-pool 192.168.2.0/24
data-dir /tmp
map 192.168.1.10 2001:db8:ca00:aaaa::c0a8:010a
```

Este arquivo possui seis parâmetros configuráveis, dois deles opcionais. A configuração que será utilizada neste exercício está apresentada a seguir. Para uma explicação da função geral dos parâmetros, reveja a experiência do NAT64.

Perceba que a regra de tradução automática do TAYGA possui como prefixo o `2001:db8:ca00:bbbb::/96`, mas que há uma regra específica de tradução com o prefixo `2001:db8:ca00:aaaa::/96`. Isto se deve ao fato de que pacotes IPv4 direcionados à Internet devem ter um prefixo diferente de pacotes direcionados a cliente do 464XLAT. Assim, qualquer pacote com destino a um IPv4 diferente de 192.168.1.10 será traduzido para o IPv6 com o prefixo `2001:db8:ca00:bbbb::/96` e será encaminhado

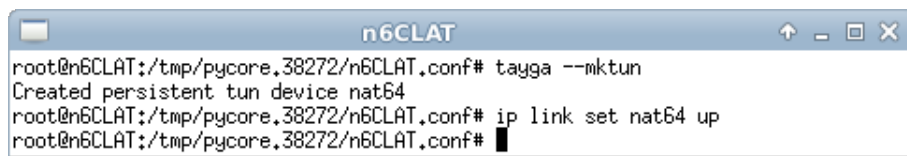
para o *gateway* IPv6 do CLAT, enquanto que um pacote IPv6 que chegar ao CLAT com destino 2001:db8:ca00:aaaa::c0a8:010a (equivalente a 2001:db8:ca00:aaaa::192.168.1.10) será traduzido para 192.168.1.10 em vez de roteado para o *gateway* IPv6, e então encaminhado para o cliente do 464XLAT.

(c) Ainda no terminal de n6CLAT, são necessárias mais algumas configurações para o funcionamento correto do TAYGA. Para isto execute os seguintes comandos:

i. Crie a interface TUN:

```
# tayga --mktun
# ip link set nat64 up
```

O resultado dos comandos é representado pela Figura 4.61.



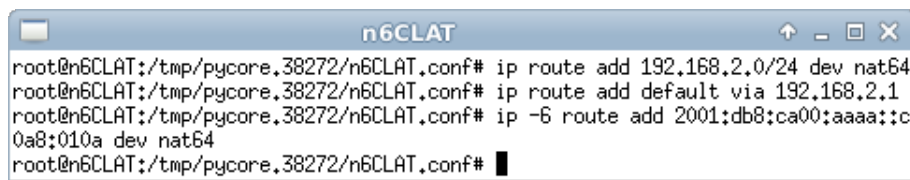
```
n6CLAT
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# tayga --mktun
Created persistent tun device nat64
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# ip link set nat64 up
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# █
```

Figura 4.61: criação da interface TUN.

ii. Configure o roteamento dos pacotes a serem traduzidos para a interface TUN:

```
# ip route add 192.168.2.0/24 dev nat64
# ip route add default via 192.168.2.1
# ip -6 route add 2001:db8:ca00:aaaa::c0a8:010a dev nat64
```

O resultado obtido deve ser similar ao apresentado na Figura 4.62.



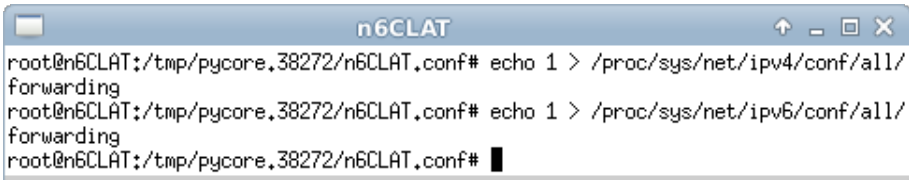
```
n6CLAT
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# ip route add 192.168.2.0/24 dev nat64
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# ip route add default via 192.168.2.1
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# ip -6 route add 2001:db8:ca00:aaaa::c0a8:010a dev nat64
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# █
```

Figura 4.62: configuração das rotas do túnel.

- iii. Habilite o encaminhamento de pacotes no *kernel* do Linux, por meio dos parâmetros do `sysctl`:

```
# echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
# echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
```

O resultado obtido deve ser similar ao apresentado na Figura 4.63.



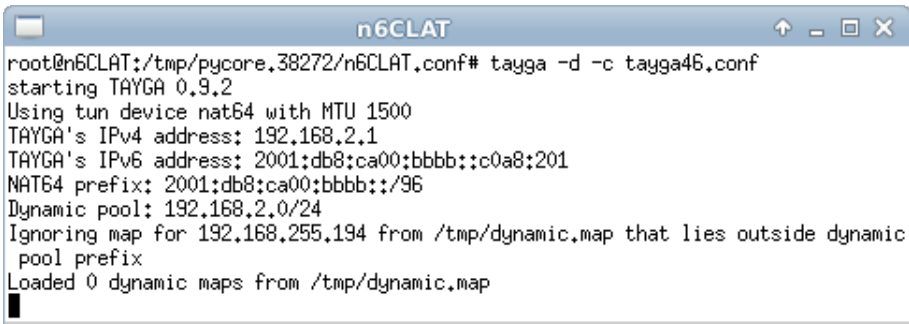
```
n6CLAT
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# █
```

Figura 4.63: *habilitando o encaminhamento de pacotes no kernel do Linux.*

- iv. Inicie o TAYGA em modo de *debug* e utilize as configurações do arquivo `tayga46.conf` descrito anteriormente:

```
# tayga -d -c tayga46.conf
```

O resultado da saída deste comando será similar ao apresentado na Figura 4.64.



```
n6CLAT
root@n6CLAT:/tmp/pycore.38272/n6CLAT.conf# tayga -d -c tayga46.conf
starting TAYGA 0.9.2
Using tun device nat64 with MTU 1500
TAYGA's IPv4 address: 192.168.2.1
TAYGA's IPv6 address: 2001:db8:ca00:bbbb::c0a8:201
NAT64 prefix: 2001:db8:ca00:bbbb::/96
Dynamic pool: 192.168.2.0/24
Ignoring map for 192.168.255.194 from /tmp/dynamic.map that lies outside dynamic pool prefix
Loaded 0 dynamic maps from /tmp/dynamic.map
█
```

Figura 4.64: *inicialização do TAYGA.*

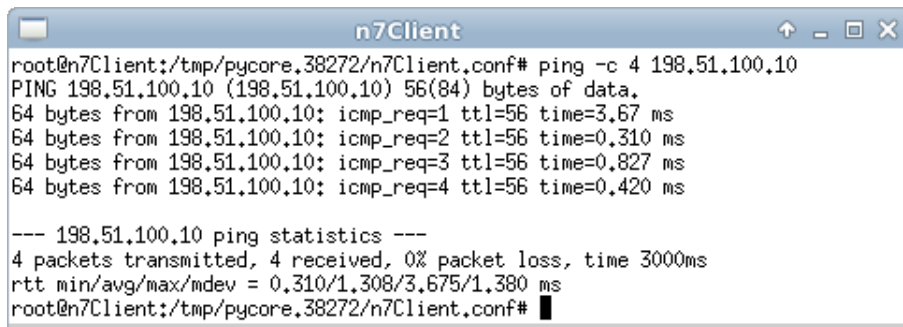
Após iniciado o TAYGA, verifique as mensagens geradas por ele em modo de *debug*.

7. Em paralelo, efetue:
- A coleta dos pacotes trafegados na interface `nat64` de `n1PLAT`, utilizando o comando `tcpdump`. As instruções de coleta de pacotes utilizando `tcpdump` se encontram no Apêndice C.

- (b) A coleta dos pacotes trafegados na interface `nat64` de `n6CLAT`, utilizando o comando `tcpdump`. As instruções de coleta de pacotes utilizando `tcpdump` se encontram no Apêndice C.
- (c) A verificação de conectividade IPv6 entre `n7Client` e `n3Host`. Para isto utilize o comando:

```
# ping -c 4 198.51.100.10
```

O resultado do comando é representado pela Figura 4.65.



```

root@n7Client:/tmp/pycore.38272/n7Client.conf# ping -c 4 198.51.100.10
PING 198.51.100.10 (198.51.100.10) 56(84) bytes of data.
64 bytes from 198.51.100.10: icmp_req=1 ttl=56 time=3.67 ms
64 bytes from 198.51.100.10: icmp_req=2 ttl=56 time=0.310 ms
64 bytes from 198.51.100.10: icmp_req=3 ttl=56 time=0.827 ms
64 bytes from 198.51.100.10: icmp_req=4 ttl=56 time=0.420 ms

--- 198.51.100.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.310/1.308/3.675/1.380 ms
root@n7Client:/tmp/pycore.38272/n7Client.conf# █

```

Figura 4.65: verificação de conectividade IPv4 de um cliente do 464XLAT.

8. Analise as capturas de pacotes feitas.
- (a) Efetue a análise dos pacotes capturados em `n1PLAT`, conforme representado na Figura 4.66.

Esta captura de pacotes na interface TUN do TAYGA oferece uma visão bastante didática do processo de tradução do 464XLAT, pois mostra os pacotes indo e retornando da Internet tanto em IPv4 quanto em IPv6, isto é; o *echo request* chega em IPv6, é traduzido para um outro *echo request* em IPv4 e encaminhado para a Internet. Quando retorna da mesma, surge na interface um *echo reply* em IPv4 que é sucedido imediatamente por um *echo reply* em IPv6, isto é; o pacote IPv4 foi traduzido de volta para IPv6 e encaminhado para a interface de rede do PLAT, que está em conexão direta com a rede do ISP. Perceba os endereços de origem e destino de todos os pacotes e veja a equivalência entre eles em IPv4 e IPv6.

Figura 4.66: captura de pacotes realizada na interface nat64 do PLAT.

- (b) Efetue a análise dos pacotes capturados em n6CLAT, conforme representado na Figura 4.67.

Nesta captura o mesmo processo de geração duplicada de pacotes de ping é observado quase instantaneamente. A diferença está na ordem dos pacotes: primeiro surgem os pacotes IPv4 e depois a sua tradução para IPv6. Ao retornar da Internet, os pacotes chegam à interface de rede do CLAT primeiramente em IPv6 e, então, são traduzidos para IPv4 e reencaminhados para a interface de rede em conexão direta como cliente do 464XLAT. Confirme a mesma equivalência de endereços de origem e destino dos pacotes em IPv6 e IPv4.

The screenshot displays the Wireshark interface with a packet capture of ICMP Echo (ping) requests and replies. The main pane shows a list of 16 packets, alternating between requests and replies. The detailed view pane shows the structure of the first packet: Internet Protocol Version 4 and Internet Control Message Protocol.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.10	198.51.100.10	ICMP	84	Echo (ping) request id=0x0021, seq=0
2	0.000409	2001:db8:ca00:aaaa::c0a8:10a	2001:db8:ca00:bbbb::c633:640a	ICMPv6	104	Echo (ping) request id=0x0021, seq=0
3	0.001555	2001:db8:ca00:bbbb::c633:640;2001:db8:ca00:aaaa::c0a8:10a	2001:db8:ca00:aaaa::c0a8:10a	ICMPv6	104	Echo (ping) reply id=0x0021, seq=1
4	0.002442	198.51.100.10	192.168.1.10	ICMP	84	Echo (ping) reply id=0x0021, seq=1
5	1.002009	192.168.1.10	198.51.100.10	ICMP	84	Echo (ping) request id=0x0021, seq=2
6	1.002107	2001:db8:ca00:aaaa::c0a8:10a	2001:db8:ca00:bbbb::c633:640a	ICMPv6	104	Echo (ping) request id=0x0021, seq=2
7	1.002520	2001:db8:ca00:bbbb::c633:640;2001:db8:ca00:aaaa::c0a8:10a	2001:db8:ca00:aaaa::c0a8:10a	ICMPv6	104	Echo (ping) reply id=0x0021, seq=2
8	1.002642	198.51.100.10	192.168.1.10	ICMP	84	Echo (ping) reply id=0x0021, seq=2
9	2.002281	192.168.1.10	198.51.100.10	ICMP	84	Echo (ping) request id=0x0021, seq=3
10	2.002416	2001:db8:ca00:aaaa::c0a8:10a	2001:db8:ca00:bbbb::c633:640a	ICMPv6	104	Echo (ping) request id=0x0021, seq=3
11	2.002767	2001:db8:ca00:bbbb::c633:640;2001:db8:ca00:aaaa::c0a8:10a	2001:db8:ca00:aaaa::c0a8:10a	ICMPv6	104	Echo (ping) reply id=0x0021, seq=3
12	2.002903	198.51.100.10	192.168.1.10	ICMP	84	Echo (ping) reply id=0x0021, seq=3
13	3.002400	192.168.1.10	198.51.100.10	ICMP	84	Echo (ping) request id=0x0021, seq=4
14	3.002883	2001:db8:ca00:aaaa::c0a8:10a	2001:db8:ca00:bbbb::c633:640a	ICMPv6	104	Echo (ping) request id=0x0021, seq=4
15	3.003690	2001:db8:ca00:bbbb::c633:640;2001:db8:ca00:aaaa::c0a8:10a	2001:db8:ca00:aaaa::c0a8:10a	ICMPv6	104	Echo (ping) reply id=0x0021, seq=4
16	3.003812	198.51.100.10	192.168.1.10	ICMP	84	Echo (ping) reply id=0x0021, seq=4

Frame 1: 84 bytes on wire (672 bits), 84 bytes captured (672 bits)
 Raw packet data
 Internet Protocol Version 4, Src: 192.168.1.10 (192.168.1.10), Dst: 198.51.100.10 (198.51.100.10)
 Internet Control Message Protocol

File: "/tmp/clat-capture.pcap" 1784 ... Packets: 16 Displayed: 16 Marked: 0 Load time: 0:00.000 Profile: Default

Figura 4.67: captura de pacotes realizada na interface nat64 do roteador CLAT.

9. Encerre a simulação, conforme descrito no Apêndice B.

Capítulo 5

Roteamento

Experiência 5.1. OSPFv3: configuração de uma única área

Objetivo

O principal objetivo deste laboratório é apresentar o funcionamento do protocolo de roteamento OSPFv3 em uma rede IPv6. Para tanto será utilizada a aplicação Quagga, que permite a configuração e utilização de protocolos de roteamento em servidores Linux. Este primeiro laboratório apresenta um cenário simples que nos permitirá conhecer o funcionamento e as configurações básicas do protocolo OSPFv3, trabalhando com uma única área.

Para a realização deste exercício será utilizada a topologia descrita no arquivo: **5-01-OSPFv3.imn**.

Introdução teórica

O OSPF é um protocolo de roteamento interno (IGP – *Interior Gateway Protocol*), que permite aos roteadores a troca de informações sobre as rotas que estes conhecem e sobre os estados dos enlaces aos quais estão conectados. A partir destas informações, o roteador mapeia a rede com o intuito de determinar a árvore de caminhos mais curtos para todas as

suas sub-redes, tendo o próprio nó como raiz. Ele utiliza o algoritmo de Dijkstra para a escolha do melhor caminho e permite a construção de topologias de rede hierárquicas, agrupando os roteadores de uma rede em áreas.

A cada uma dessas áreas é atribuído um identificador único (Area-ID) de 32 *bits* e todos os roteadores de uma mesma área mantêm um banco de dados de estado separado, de modo que a topologia de uma área seja desconhecida fora dela. Isto reduz a quantidade de tráfego de roteamento entre diferentes partes da rede. A área de *backbone* é a responsável por distribuir as informações de roteamento e tem que ser identificada pelo ID 0 (ou 0.0.0.0). Em uma rede em que não existem tais divisões, a área de *backbone* é a única a ser configurada.

O OSPFv3 é um protocolo utilizado unicamente em redes IPv6 e foi baseado na versão do OSPFv2, utilizada em redes IPv4. Deste modo, em uma rede com pilha dupla é necessário utilizar tanto OSPFv2, para o roteamento IPv4, quanto o OSPFv3, para o roteamento IPv6. Mais informações podem ser obtidas na RFC 5340 (Coltun *et al.*, 2008).

Roteiro experimental

1. Inicie o CORE e abra o arquivo **5-01-OSPFv3.imn** localizado no diretório lab, dentro do Desktop. A topologia de rede, representada pela Figura 5.1, deve aparecer.

Esta topologia apresenta uma rede local composta por dois nós conectados a sub-redes diferentes e por três roteadores, nos quais será configurado o protocolo de roteamento dinâmico OSPFv3.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 e IPv4 nos nós n1Backbone, n2Backbone, n3Backbone, n4HostA, e n5HostB.

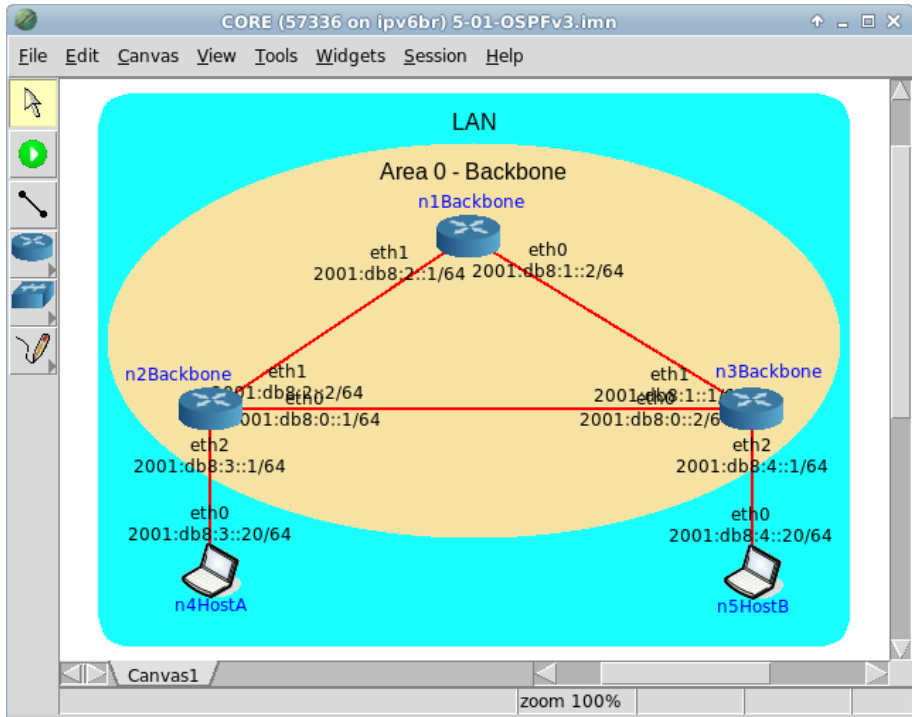


Figura 5.1: topologia da Experiência 5.1 no CORE.

3. Verifique a conectividade entre os dispositivos da rede:
 - (a) Acesse o terminal da máquina n4HostA com um duplo-clique e digite o seguinte comando:

```
# ping6 -c 4 2001:db8:3::1
```

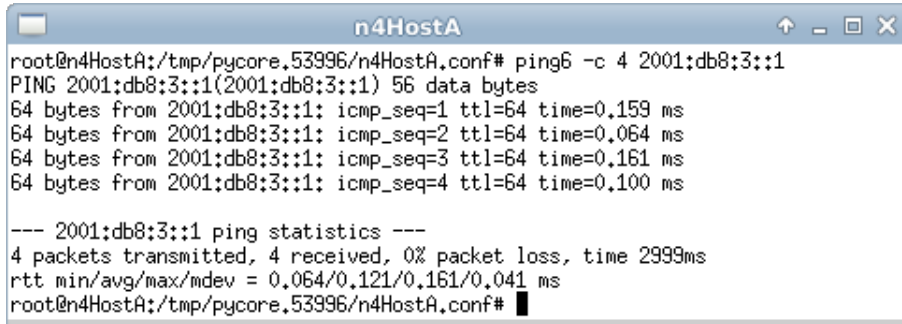
O resultado do comando é representado pela Figura 5.2.

A comunicação pôde ser estabelecida pois o endereço utilizado pertence ao roteador n2Backbone, que está diretamente conectado ao n4HostA.

- (b) A seguir, teste a conectividade entre o n4HostA e o n5HostB. Ainda no terminal do n4HostA, digite o seguinte comando:

```
# ping6 -c 4 2001:db8:4::20
```

O resultado do comando é representado pela Figura 5.3.



```

n4HostA
root@n4HostA:/tmp/pycore.53996/n4HostA.conf# ping6 -c 4 2001:db8:3::1
PING 2001:db8:3::1(2001:db8:3::1) 56 data bytes
64 bytes from 2001:db8:3::1: icmp_seq=1 ttl=64 time=0,159 ms
64 bytes from 2001:db8:3::1: icmp_seq=2 ttl=64 time=0,064 ms
64 bytes from 2001:db8:3::1: icmp_seq=3 ttl=64 time=0,161 ms
64 bytes from 2001:db8:3::1: icmp_seq=4 ttl=64 time=0,100 ms

--- 2001:db8:3::1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0,064/0,121/0,161/0,041 ms
root@n4HostA:/tmp/pycore.53996/n4HostA.conf# █

```

Figura 5.2: teste de conectividade entre n4HostA e n2Backbone.



```

n4HostA
root@n4HostA:/tmp/pycore.53997/n4HostA.conf# ping6 -c 4 2001:db8:4::20
PING 2001:db8:4::20(2001:db8:4::20) 56 data bytes
From 2001:db8:3::1 icmp_seq=1 Destination unreachable: No route
From 2001:db8:3::1 icmp_seq=2 Destination unreachable: No route
From 2001:db8:3::1 icmp_seq=3 Destination unreachable: No route
From 2001:db8:3::1 icmp_seq=4 Destination unreachable: No route

--- 2001:db8:4::20 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3016ms
root@n4HostA:/tmp/pycore.53997/n4HostA.conf# █

```

Figura 5.3: teste de conectividade entre n4HostA e n5HostB.

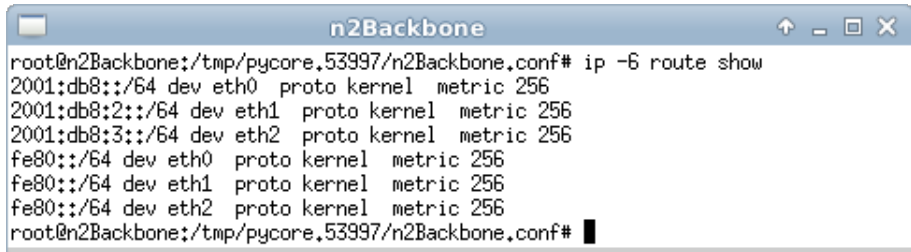
Como estes dois equipamentos não estão diretamente conectados, eles dependem da comunicação entre os roteadores da rede para que haja conectividade. No entanto, mesmo com os roteadores n2Backbone e n3Backbone interligados fisicamente, não houve comunicação.

- (c) Abra o terminal do roteador n2Backbone com um duplo-clique sobre ele e digite o seguinte comando:

```
# ip -6 route show
```

O resultado do comando é representado pela Figura 5.4.

Este comando mostra todas as redes conhecidas pelo roteador n2Backbone. Como é possível observar, ele também só conhece as redes que estão diretamente conectadas a ele. Por isso, não consegue alcançar os dispositivos que estão na rede 2001:db8:4::/64. Repita este teste nos outros dois roteadores e confirme se eles apresentam o mesmo comportamento.



```

n2Backbone
root@n2Backbone:/tmp/pycore.53997/n2Backbone.conf# ip -6 route show
2001:db8::/64 dev eth0 proto kernel metric 256
2001:db8:2::/64 dev eth1 proto kernel metric 256
2001:db8:3::/64 dev eth2 proto kernel metric 256
fe80::/64 dev eth0 proto kernel metric 256
fe80::/64 dev eth1 proto kernel metric 256
fe80::/64 dev eth2 proto kernel metric 256
root@n2Backbone:/tmp/pycore.53997/n2Backbone.conf# █

```

Figura 5.4: tabela de rotas do roteador n2Backbone.

Para proporcionar conectividade entre todos os dispositivos da rede, uma possibilidade seria configurar rotas estáticas entre os roteadores, *ensinando-lhes* todos os caminhos necessários. Porém, caso houvesse alguma mudança nos *links* da rede, seria necessário reconfigurar tudo manualmente.

Para automatizar esta tarefa, pode-se trabalhar com protocolos de roteamento dinâmico. Com o uso destes protocolos os roteadores trocam mensagens entre si divulgando as rotas que eles conhecem e informando qualquer mudança de estado dos *links* da rede.

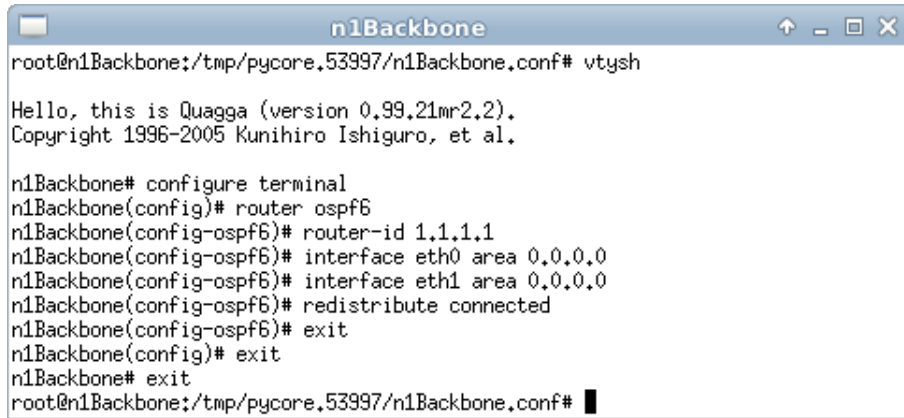
4. Será utilizado para configurar as rotas entre as redes IPv6 o protocolo de roteamento interno OSPFv3. Configure-o nos três roteadores presentes na topologia.
 - (a) Abra o terminal do roteador n1Backbone. Será utilizada a aplicação Quagga para configurar o OSPFv3. Para tanto, digite os seguintes comandos:

```

# vtysh
# configure terminal
# router ospf6
# router-id 1.1.1.1
# interface eth0 area 0.0.0.0
# interface eth1 area 0.0.0.0
# redistribute connected
# exit
# exit
# exit

```

O resultado do comando é representado pela Figura 5.5.

A terminal window titled 'n1Backbone' showing the configuration of a Quagga router. The prompt is 'root@n1Backbone:/tmp/pycore.53997/n1Backbone.conf#'. The user enters 'vtysh', which shows the Quagga version (0.99.21mr2.2) and copyright information. Then, the user enters 'configure terminal', 'router ospf6', 'router-id 1.1.1.1', 'interface eth0 area 0.0.0.0', 'interface eth1 area 0.0.0.0', 'redistribute connected', and 'exit' multiple times to return to the root prompt.

```
root@n1Backbone:/tmp/pycore.53997/n1Backbone.conf# vtysh
Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n1Backbone# configure terminal
n1Backbone(config)# router ospf6
n1Backbone(config-ospf6)# router-id 1.1.1.1
n1Backbone(config-ospf6)# interface eth0 area 0.0.0.0
n1Backbone(config-ospf6)# interface eth1 area 0.0.0.0
n1Backbone(config-ospf6)# redistribute connected
n1Backbone(config-ospf6)# exit
n1Backbone(config)# exit
n1Backbone# exit
root@n1Backbone:/tmp/pycore.53997/n1Backbone.conf#
```

Figura 5.5: comandos de configuração do roteador n1Backbone.

Estes comandos realizam as seguintes funções:

vtysh

Acessa a interface de configuração do Quagga.

configure terminal

Acessa o modo de edição do Quagga, interface utilizada para definir as configurações dos protocolos de roteamento.

router ospf6

Indica que será configurado o protocolo OSPF para IPv6.

router-id 1.1.1.1

Adiciona um número de 32 *bits* para identificar o roteador.

redistribute connected

Configura o roteador para informar aos outros roteadores da rede as rotas diretamente conectadas que ele conhece.

interface eth0 area 0.0.0.0

Adiciona a interface eth0 do roteador à área *backbone* do OSPF, de modo que todas as informações de rotas sejam divulgadas aos roteadores que estiverem conectados a esta interface.

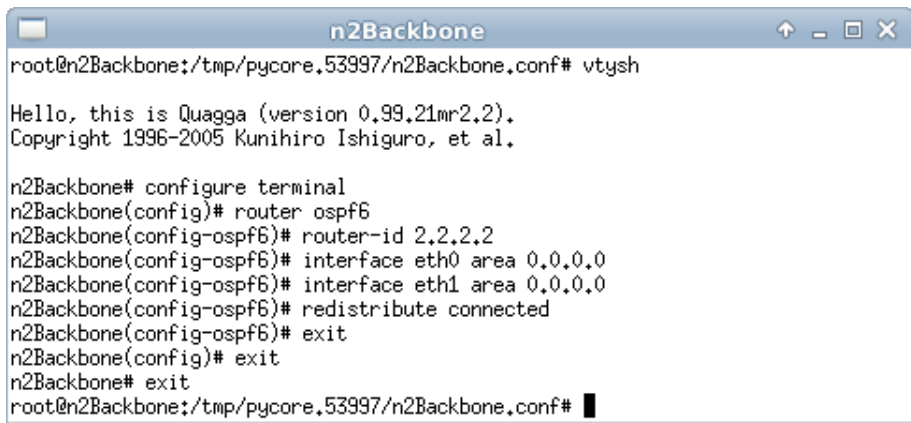
interface eth1 area 0.0.0.0

Do mesmo modo que no item anterior, adiciona a interface eth1 à área *backbone* do OSPF.

- (b) Configure também o roteador n2Backbone. Para isto, abra o terminal do roteador n2Backbone e digite os seguintes comandos:

```
# vtysh
# configure terminal
# router ospf6
# router-id 2.2.2.2
# interface eth0 area 0.0.0.0
# interface eth1 area 0.0.0.0
# redistribute connected
# exit
# exit
# exit
```

O resultado do comando é representado pela Figura 5.6.



```
n2Backbone
root@n2Backbone:/tmp/pycore.53997/n2Backbone.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

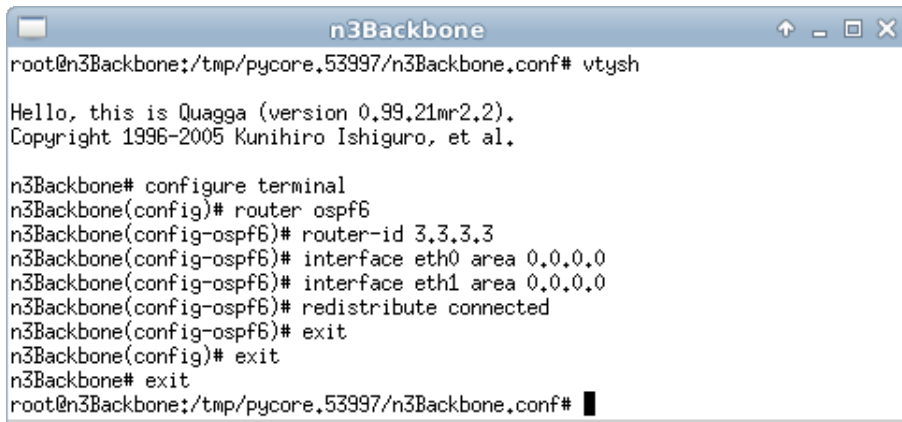
n2Backbone# configure terminal
n2Backbone(config)# router ospf6
n2Backbone(config-ospf6)# router-id 2.2.2.2
n2Backbone(config-ospf6)# interface eth0 area 0.0.0.0
n2Backbone(config-ospf6)# interface eth1 area 0.0.0.0
n2Backbone(config-ospf6)# redistribute connected
n2Backbone(config-ospf6)# exit
n2Backbone(config)# exit
n2Backbone# exit
root@n2Backbone:/tmp/pycore.53997/n2Backbone.conf# █
```

Figura 5.6: comandos de configuração do roteador n2Backbone.

- (c) Por fim, configure do mesmo modo o roteador n3Backbone. Para isto, abra o terminal do roteador n3Backbone com um duplo-clique e digite os seguintes comandos:

```
# vtysh
# configure terminal
# router ospf6
# router-id 3.3.3.3
# interface eth0 area 0.0.0.0
# interface eth1 area 0.0.0.0
# redistribute connected
# exit
# exit
# exit
```

O resultado do comando é representado pela Figura 5.7.



```
n3Backbone
root@n3Backbone:/tmp/pycore.53997/n3Backbone.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n3Backbone# configure terminal
n3Backbone(config)# router ospf6
n3Backbone(config-ospf6)# router-id 3.3.3.3
n3Backbone(config-ospf6)# interface eth0 area 0.0.0.0
n3Backbone(config-ospf6)# interface eth1 area 0.0.0.0
n3Backbone(config-ospf6)# redistribute connected
n3Backbone(config-ospf6)# exit
n3Backbone(config)# exit
n3Backbone# exit
root@n3Backbone:/tmp/pycore.53997/n3Backbone.conf# █
```

Figura 5.7: comandos de configuração do roteador n3Backbone.

5. Valide as configurações utilizando inicialmente comandos do Quagga.
- (a) Abra o terminal do roteador `n2Backbone` e digite os seguintes comandos:

```
# vtysh
# show ipv6 ospf6
# show ipv6 ospf6 neighbor
# show ipv6 route
# exit
```

O resultado do comando é representado pela Figura 5.8.

Estes comandos apresentam as seguintes informações:

```
vttysh
```

Acessa a interface de configuração do Quagga.

```
show ipv6 ospf6
```

Mostra um resumo das informações do processo do OSPF. Entre elas, há quanto tempo o processo está rodando, qual o ID, a quais áreas o roteador está conectado e quais interfaces estão conectadas a cada área.

```
show ipv6 ospf6 neighbor
```

Mostra um resumo das informações dos vizinhos OSPF. Indica o ID do vizinho, se a sessão está estabelecida, há quanto tempo, etc.

```
show ipv6 route
```

Mostra a tabela de roteamento IPv6. As rotas marcadas com a letra "o" são as aprendidas por meio do protocolo OSPF.

- (b) Verifique também as rotas que estão presentes no Sistema Operacional do roteador. Ainda no terminal do roteador `n2Backbone`, digite o seguinte comando:

```
# ip -6 route show
```

O resultado do comando é representado pela Figura 5.9.

Compare com o resultado do teste feito no passo 3 deste exercício e veja que agora `n2Backbone` conhece todas as redes existentes na topologia.

Repita os mesmos comandos nos outros dois roteadores e compare os resultados. Eles devem ser similares, alterando apenas o ID dos vizinhos.

```

n2Backbone
root@n2Backbone:/tmp/pycore.53997/n2Backbone.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n2Backbone# show ipv6 ospf6
 OSPFv3 Routing Process (0), Instance ID 0, Router-ID 2.2.2.2
 Running 00:54:27
 Number of AS scoped LSAs is 2
 Number of areas in this router is 1
 Area 0,0,0,0
   Number of Area scoped LSAs is 12
   Interface attached to this area: eth0 eth1
n2Backbone# show ipv6 ospf6 neighbor
Neighbor ID      Pri   DeadTime  State/IFState      Duration I/F[State]
3.3.3.3          1     00:00:34  Full/BDR           00:53:11 eth0[DR]
1.1.1.1          1     00:00:32  Full/DR            00:54:28 eth1[BDR]
n2Backbone# show ipv6 route
Codes: K - kernel route, C - connected, S - static, R - RIPng,
       o - OSPF6, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

C>* ::1/128 is directly connected, lo
o 2001:db8::/64 [110/1] is directly connected, eth0, 00:54:54
C>* 2001:db8::/64 is directly connected, eth0
o>* 2001:db8:1::/64 [110/2] via fe80::200:ff:feaa:4, eth1, 00:53:14
o 2001:db8:2::/64 [110/1] is directly connected, eth1, 00:54:42
C>* 2001:db8:2::/64 is directly connected, eth1
C>* 2001:db8:3::/64 is directly connected, eth2
o>* 2001:db8:4::/64 [110/2] via fe80::200:ff:feaa:1, eth0, 00:52:40
C * fe80::/64 is directly connected, eth2
C * fe80::/64 is directly connected, eth1
C>* fe80::/64 is directly connected, eth0
n2Backbone# exit
root@n2Backbone:/tmp/pycore.53997/n2Backbone.conf# █

```

Figura 5.8: status do processo do OSPFv3, vizinhança e tabela de rotas do roteador n2Backbone.

```

n2Backbone
root@n2Backbone:/tmp/pycore.53997/n2Backbone.conf# ip -6 route show
2001:db8::/64 dev eth0 proto kernel metric 256
2001:db8:1::/64 via fe80::200:ff:feaa:4 dev eth1 proto zebra metric 2
2001:db8:2::/64 dev eth1 proto kernel metric 256
2001:db8:3::/64 dev eth2 proto kernel metric 256
2001:db8:4::/64 via fe80::200:ff:feaa:1 dev eth0 proto zebra metric 2
fe80::/64 dev eth0 proto kernel metric 256
fe80::/64 dev eth1 proto kernel metric 256
fe80::/64 dev eth2 proto kernel metric 256
root@n2Backbone:/tmp/pycore.53997/n2Backbone.conf# █

```

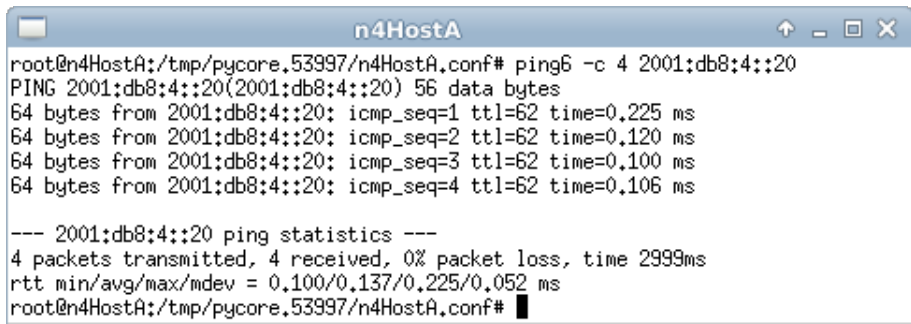
Figura 5.9: tabela de rotas do roteador n2Backbone.

- Para finalizar, repita o teste de conectividade entre o n4HostA e o n5HostB e veja se agora é possível realizar a conexão.

- (a) Abra o terminal da máquina n4HostA e digite o seguinte comando:

```
# ping6 -c 4 2001:db8:4::20
```

O resultado do comando é representado pela Figura 5.10.



```
n4HostA
root@n4HostA:/tmp/pycore.53997/n4HostA.conf# ping6 -c 4 2001:db8:4::20
PING 2001:db8:4::20(2001:db8:4::20) 56 data bytes
64 bytes from 2001:db8:4::20: icmp_seq=1 ttl=62 time=0.225 ms
64 bytes from 2001:db8:4::20: icmp_seq=2 ttl=62 time=0.120 ms
64 bytes from 2001:db8:4::20: icmp_seq=3 ttl=62 time=0.100 ms
64 bytes from 2001:db8:4::20: icmp_seq=4 ttl=62 time=0.106 ms

--- 2001:db8:4::20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.100/0.137/0.225/0.052 ms
root@n4HostA:/tmp/pycore.53997/n4HostA.conf# █
```

Figura 5.10: teste de conectividade entre n4HostA e n5HostB.

Após a configuração do OSPFv3, a conectividade entre todos os dispositivos da rede pode ser estabelecida.

Analise o funcionamento do protocolo OSPFv3 desabilitando as interfaces dos roteadores e verificando se novos caminhos são aprendidos para se conectarem a determinados segmentos da rede. Por exemplo, desabilite a interface eth0 do roteador n2Backbone e verifique se a conectividade entre as máquinas n4HostA e n5HostB continua ativa. Analese também as tabelas de roteamento para verificar qual caminho passou a ser utilizado no estabelecimento da conexão ou utilize o comando `traceroute6` para realizar estes testes e compare os caminhos utilizados.

7. Encerre a simulação, conforme descrito no Apêndice B.

Experiência 5.2. BGP

Objetivo

O principal objetivo desse laboratório é apresentar o funcionamento do protocolo de roteamento BGP em uma rede IPv6. Para tanto, será utilizada a aplicação Quagga, que permite a configuração e utilização de protocolos de roteamento em servidores Linux.

Na primeira parte da experiência serão apresentados exemplos de configurações básicas para o funcionamento do protocolo BGP em um cenário composto por roteadores que representam os Sistemas Autônomos (AS) da Internet, sendo possível observar o estabelecimento das sessões iBGP e eBGP. A segunda parte da experiência apresenta exemplos de configurações relacionadas ao estabelecimento de políticas de roteamento, as quais permitem influenciar o tráfego de entrada e de saída do AS.

Para a realização deste exercício será utilizada a topologia descrita no arquivo: **5-02-BGP.imn**.

Introdução teórica

A Internet foi projetada para ser uma rede resiliente, ou seja, para resistir a falhas no sistema de comunicação, encontrando caminhos *alternativos* entre os pontos de origem e de destino. Esta característica é garantida pelo fato da Internet ter uma estrutura descentralizada, formada pela interligação de diversas redes.

As redes que compõem a Internet são chamadas de Sistemas Autônomos ou simplesmente AS (do inglês *Autonomous Systems*). Elas podem ser provedores de trânsito, provedores de serviços ou simplesmente usuários finais. Os ASs são identificados por um número chamado ASN (*Autonomous System Number*), que pode ser de 16 ou 32 *bits*.

O *Border Gateway Protocol* versão 4 (BGP-4) é hoje o protocolo EGP padrão da Internet, utilizado para fazer o roteamento de pacotes IP entre os ASs. Definido pela RFC 4271 (Rekhter *et al.*, 2006), o BGP é um protocolo do tipo *path vector*, que apresenta como principais vantagens a alta escalabilidade e a diversidade de atributos utilizados para estabelecimento de políticas de roteamento.

Para habilitar o roteamento entre ASs por meio do IPv6, é preciso habilitar as extensões multiprotocolo do BGP. O MP-BGP ou *multiprotocol BGP*, é uma extensão do BGP-4 definida com o intuito de torná-lo capaz de carregar informações de roteamento de múltiplas famílias de protocolos da Camada de Rede, por exemplo, IPv6, IPX, L3VPN, etc. O suporte a essa extensão é obrigatório para se realizar o roteamento externo IPv6, visto que não há uma versão específica de BGP para tratar esta tarefa.

A palavra *autônomo*, da expressão Sistema Autônomo, indica que o AS possui autonomia para definir sua políticas de roteamento, influenciando a forma como o tráfego vindo da Internet chega à sua rede e como o tráfego da sua rede sai com destino a outras redes na Internet.

As políticas de roteamento dividem-se em políticas de entrada e políticas de saída, e são definidas baseadas nas informações de roteamento trocadas entre os ASs da Internet, ou seja, nos prefixos anunciados e recebidos pelo AS.

As políticas de entrada, chamadas de AS-IN, são aplicadas sobre as informações aprendidas de outros ASs e influenciam o tráfego de saída do AS. As políticas de saída, chamadas de AS-OUT, são aplicadas sobre as informações de roteamento que um AS divulga para a Internet e influenciam o tráfego de entrada do AS. A Figura 5.11 ilustra este fluxo entre as trocas de informações de roteamento e sua influência na direção do tráfego.

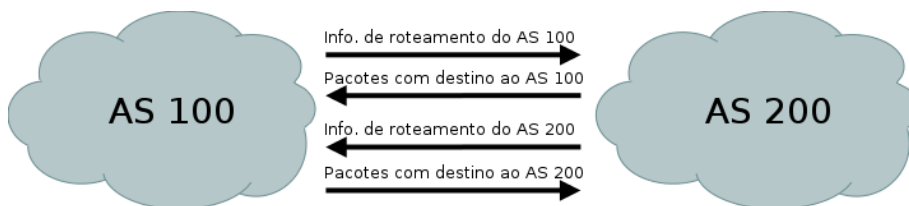


Figura 5.11: a direção do anúncio é oposta à direção do tráfego que ele influencia.

Roteiro experimental

1. Inicie o CORE e abra o arquivo **5-02-BGP.imn** localizado no diretório `lab`, dentro do Desktop. A topologia de rede, representada pela Figura 5.12, deve aparecer.

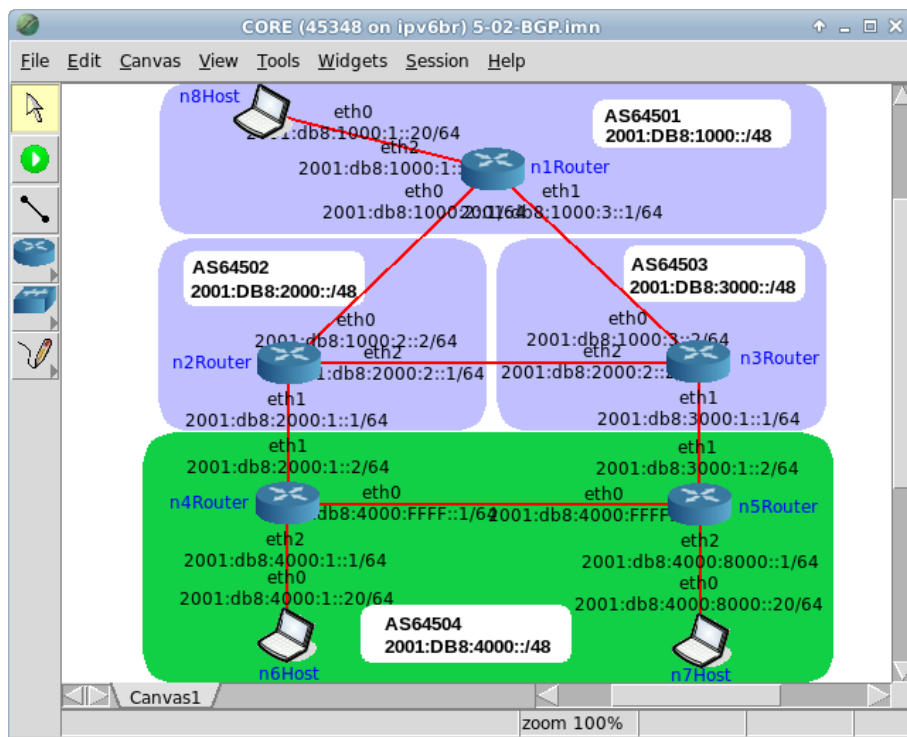


Figura 5.12: topologia da Experiência 5.2 no CORE.

2. Conforme descrito nos Apêndices B e C, inicialize a simulação, verifique a configuração de endereços IPv6 e IPv4 nos nós n1Router, n2Router, n3Router, n4Router, n5Router, n6Host, n7Host e n8Host.
 3. Verifique a conectividade entre os dispositivos da rede:
- (a) Acesse o terminal da máquina n6Host com um duplo-clique e digite o seguinte comando:

```
# ping6 -c 4 2001:db8:1000:1::20
```

O resultado do comando é representado pela Figura 5.13.



```

root@n6Host:/tmp/pycore.54148/n6Host.conf# ping6 -c 4 2001:db8:1000:1::20
PING 2001:db8:1000:1::20(2001:db8:1000:1::20) 56 data bytes
From 2001:db8:4000:1::1 icmp_seq=1 Destination unreachable: No route
From 2001:db8:4000:1::1 icmp_seq=2 Destination unreachable: No route
From 2001:db8:4000:1::1 icmp_seq=3 Destination unreachable: No route
From 2001:db8:4000:1::1 icmp_seq=4 Destination unreachable: No route

--- 2001:db8:1000:1::20 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3001ms
root@n6Host:/tmp/pycore.54148/n6Host.conf# █

```

Figura 5.13: teste de conectividade entre dois mboxes de ASs diferentes.

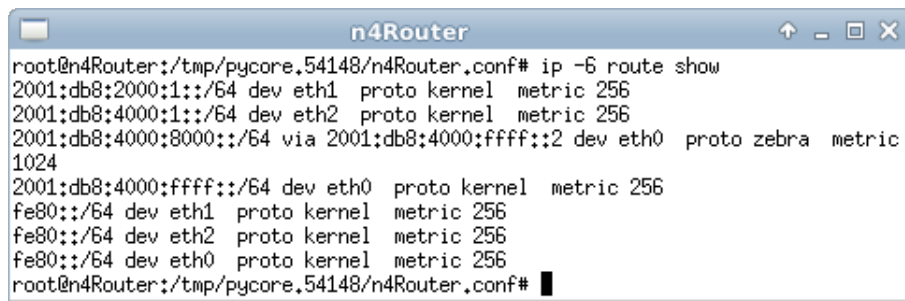
A comunicação entre n6Host, do AS64504, e n8Host, do AS64501, não pode ser estabelecida porque estes dois equipamentos não estão diretamente conectados, dependendo da comunicação entre os roteadores das redes para que haja conectividade. No entanto, não há protocolos de roteamento dinâmico ou rotas estáticas configurados nos roteadores do AS64504.

- (b) Abra o terminal do roteador n4Router e digite o seguinte comando:

```
# ip -6 route show
```

O resultado do comando é representado pela Figura 5.14.

Este comando mostra todas as redes conhecidas pelo roteador n4Router. Como é possível observar, ele também só conhece as redes do próprio AS e por isso não consegue alcançar os dispositivos que estão no AS64501. Repita este teste no roteador n5Router e confirme se ele apresenta o mesmo comportamento.



```

n4Router
root@n4Router:/tmp/pycore.54148/n4Router.conf# ip -6 route show
2001:db8:2000:1::/64 dev eth1 proto kernel metric 256
2001:db8:4000:1::/64 dev eth2 proto kernel metric 256
2001:db8:4000:8000::/64 via 2001:db8:4000:ffff::2 dev eth0 proto zebra metric
1024
2001:db8:4000:ffff::/64 dev eth0 proto kernel metric 256
fe80::/64 dev eth1 proto kernel metric 256
fe80::/64 dev eth2 proto kernel metric 256
fe80::/64 dev eth0 proto kernel metric 256
root@n4Router:/tmp/pycore.54148/n4Router.conf# █

```

Figura 5.14: configurações de rede iniciais de um roteador do AS64504.

Para que haja conectividade entre todos os ASs deste cenário, será preciso trabalhar com o protocolo de roteamento BGP. Por meio deste protocolo, os roteadores dos ASs trocam mensagens entre si, divulgando as melhores rotas que eles conhecem e informando qualquer mudança na conectividade entre as redes.

Neste cenário, os ASs 64502 e 64503 representam provedores de trânsito, ou seja, são as redes contratadas para fornecer acesso à Internet aos outros ASs e divulgar seus prefixos IP às outras redes. Os roteadores n2Router e n3Router já estão configurados. É preciso agora configurar os equipamentos do AS64504.

4. Inicialmente, configure o protocolo BGP entre os dois roteadores presentes no AS64504. Este tipo de sessão é chamada de iBGP, pois estabelece uma sessão BGP internamente entre roteadores de um mesmo AS. Este procedimento é importante, pois garante que todos os roteadores do AS possuam a mesma versão de tabela de roteamento BGP.
- (a) Para configurar o BGP será utilizada a aplicação Quagga. Abra o terminal do roteador n4Router e execute os seguintes comandos:

```

# vtysh
# configure terminal
# router bgp 64504
# bgp router-id 4.4.4.4
# neighbor 2001:db8:4000:FFFF::2 remote-as 64504
# neighbor 2001:db8:4000:FFFF::2 description iBGP com n5Router
# address-family ipv6

```



```
# neighbor 2001:db8:4000:FFFF::2 activate
# neighbor 2001:db8:4000:FFFF::2 next-hop-self
# exit
# exit
# exit
# exit
```

O resultado dos comandos é representado pela Figura 5.15.

```
n4Router
root@n4Router:/tmp/pycore.49979/n4Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n4Router# configure terminal
n4Router(config)# router bgp 64504
n4Router(config-router)# bgp router-id 4.4.4.4
n4Router(config-router)# neighbor 2001:db8:4000:ffff::2 remote-as 64504
n4Router(config-router)# neighbor 2001:db8:4000:ffff::2 description iBGP com n5R
outer
n4Router(config-router)# address-family ipv6
n4Router(config-router-af)# neighbor 2001:db8:4000:ffff::2 activate
n4Router(config-router-af)# neighbor 2001:db8:4000:ffff::2 next-hop-self
n4Router(config-router-af)# exit
n4Router(config-router)# exit
n4Router(config)# exit
n4Router# exit
root@n4Router:/tmp/pycore.49979/n4Router.conf# █
```

Figura 5.15: comandos de configuração do iBGP no roteador Quagga n4Router.

Estes comandos realizam as seguintes funções:

`vttysh`

Acessa a interface de configuração do Quagga.

`configure terminal`

Acessa o modo de edição do Quagga, interface utilizada para definir as configurações dos protocolos de roteamento.

`router bgp 64504`

Indica um processo BGP no AS64504.

`bgp router-id 4.4.4.4`

Indica um número de identificação do roteador.

`neighbor 2001:db8:4000:FFFF::2 remote-as 64504`

Especifica o ASN do roteador vizinho.

```
neighbor 2001:db8:4000:FFFF::2 description iBGP com n5Router
```

Apresenta algum texto informativo sobre o vizinho. Apesar de não obrigatório, adicionar comentários pode facilitar a detecção de problemas na configuração de sessões BGP.

```
address-family ipv6
```

Inicia o bloco de comandos específicos de IPv6.

```
neighbor 2001:db8:4000:FFFF::2 activate
```

Ativa o vizinho para a família IPv6.

```
neighbor 2001:db8:4000:FFFF::2 next-hop-self
```

Repassa aos vizinhos iBGP que o endereço de próximo salto para alcançar as redes que estão sendo anunciadas é o endereço do próprio roteador.

- (b) Configure também o roteador n5Router. Para isto, abra o terminal do roteador n5Router e digite os seguintes comandos:

```
# vtysh
# configure terminal
# router bgp 64504
# bgp router-id 5.5.5.5
# neighbor 2001:db8:4000:FFFF::1 remote-as 64504
# neighbor 2001:db8:4000:FFFF::1 description iBGP com n4Router
# address-family ipv6
# neighbor 2001:db8:4000:FFFF::1 activate
# neighbor 2001:db8:4000:FFFF::1 next-hop-self
# exit
# exit
# exit
# exit
```

O resultado dos comandos é representado pela Figura 5.16.

Os comandos são os mesmos utilizados na configuração do roteador n4Router.

```

root@n5Router:~/tmp/pycore.49933/n5Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n5Router# configure terminal
n5Router(config)# router bgp 64504
n5Router(config-router)# bgp router-id 5.5.5.5
n5Router(config-router)# neighbor 2001:db8:4000:ffff::1 remote-as 64504
n5Router(config-router)# neighbor 2001:db8:4000:ffff::1 description iBGP com n4R
outer
n5Router(config-router)# address-family ipv6
n5Router(config-router-af)# neighbor 2001:db8:4000:ffff::1 activate
n5Router(config-router-af)# neighbor 2001:db8:4000:ffff::1 next-hop-self
n5Router(config-router-af)# exit
n5Router(config-router)# exit
n5Router(config)# exit
n5Router# exit
root@n5Router:~/tmp/pycore.49933/n5Router.conf# █

```

Figura 5.16: comandos de configuração do iBGP no roteador Quagga n5Router.

- (c) Para verificar se as configurações foram feitas corretamente, abra o terminal do roteador n4Router e digite os seguintes comandos:

```

# vtysh
# show ipv6 bgp summary
# exit

```

O resultado destes comandos é representado pela Figura 5.17.

Este comando apresenta as informações relacionadas às sessões BGP estabelecidas. Dentre os dados apresentados, há as informações específicas do roteador do AS vizinho como: seu endereço IPv6 (Neighbor); versão do protocolo BGP utilizada por ele (V); número do Sistema Autônomo (AS); quantidade de mensagens trocadas (MsgRcvd / MsgSent); versão de tabela (TblVer); se há mensagens nas filas de envio ou recebimento (InQ / OutQ); há quanto tempo a sessão está estabelecida ou interrompida (Up / Down); o estado da sessão e o número de prefixos aprendidos (State / PfxRcd).

Repita o mesmo comando no roteador n5Router.

```

root@n4Router:/tmp/pycore.49979/n4Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n4Router# show ipv6 bgp summary
BGP router identifier 4.4.4.4, local AS number 64504
RIB entries 0, using 0 bytes of memory
Peers 1, using 2524 bytes of memory

Neighbor      V   AS MsgRcvd MsgSent  TblVer  InQ  OutQ Up/Down  State/PfxRcd
2001:db8:4000:ffff::2
              4 64504      3      8        0   0   0 00:00:29      0

Total number of neighbors 1
n4Router# exit
root@n4Router:/tmp/pycore.49979/n4Router.conf# █

```

Figura 5.17: exibindo a vizinhança iBGP no roteador n4Router.

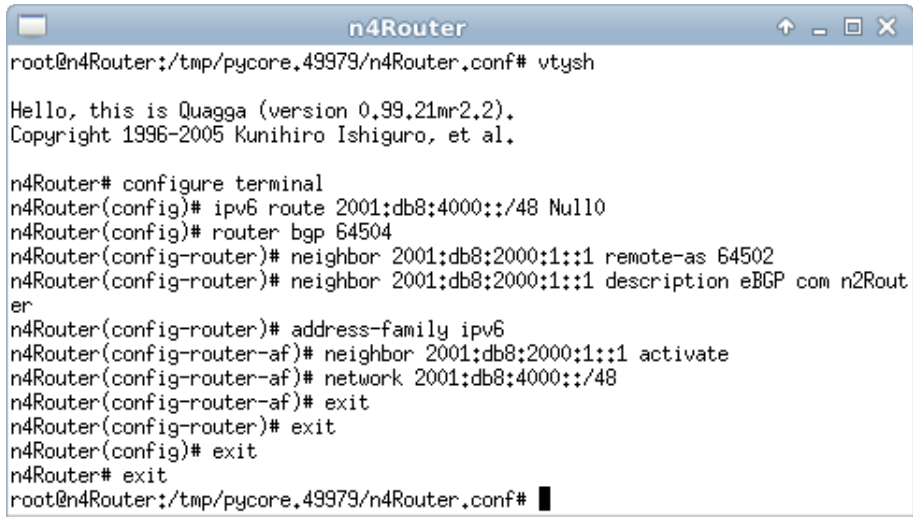
5. Em uma sessão iBGP, os roteadores divulgam somente as rotas que aprenderam externamente. Por este motivo no item anterior não havia nenhum prefixo aprendido. Para que o AS64504 conheça as redes de outros ASs e divulgue o prefixo da sua rede (2001:db8:4000::/48) é preciso estabelecer as sessões eBGP, ou seja, configurar as sessões BGP com os roteadores dos provedores de trânsito.
- (a) Abra o terminal do roteador n4Router e digite os comandos:

```

# vtysh
# configure terminal
# ipv6 route 2001:db8:4000::/48 Null0
# router bgp 64504
# neighbor 2001:db8:2000:1::1 remote-as 64502
# neighbor 2001:db8:2000:1::1 description eBGP com n2Router
# address-family ipv6
# neighbor 2001:db8:2000:1::1 activate
# network 2001:db8:4000::/48
# exit
# exit
# exit
# exit

```

O resultado será como o representado pela Figura 5.18.



```

root@n4Router:/tmp/pycore.49979/n4Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n4Router# configure terminal
n4Router(config)# ipv6 route 2001:db8:4000::/48 Null0
n4Router(config)# router bgp 64504
n4Router(config-router)# neighbor 2001:db8:2000:1::1 remote-as 64502
n4Router(config-router)# neighbor 2001:db8:2000:1::1 description eBGP com n2Router
n4Router(config-router)# address-family ipv6
n4Router(config-router-af)# neighbor 2001:db8:2000:1::1 activate
n4Router(config-router-af)# network 2001:db8:4000::/48
n4Router(config-router-af)# exit
n4Router(config-router)# exit
n4Router(config)# exit
n4Router# exit
root@n4Router:/tmp/pycore.49979/n4Router.conf# █

```

Figura 5.18: comandos de configuração do eBGP no roteador Quagga n4Router.

Os comandos são os mesmos utilizados para estabelecer uma sessão iBGP, com a diferença de que agora o vizinho pertence a outro AS e foram utilizados os seguintes comandos adicionais:

```
ipv6 route 2001:db8:4000::/48 Null0
```

Por padrão, o protocolo BGP só divulga rotas que encontram-se na tabela de rotas do roteador. Então, este comando cria uma rota estática para o prefixo que se pretende divulgar apenas para forçar a sua existência na tabela de rotas.

```
network 2001:db8:4000::/48
```

Declara a rede que se pretende divulgar.

- (b) Para estabelecer a sessão eBGP entre o roteador n5Router e o n3Router, abra o terminal do roteador n5Router e digite os seguintes comandos:

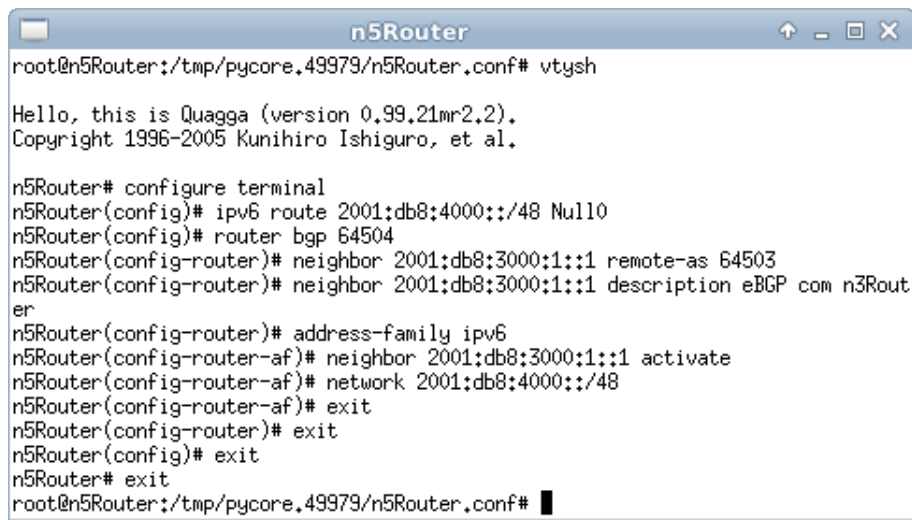
```

# vtysh
# configure terminal
# ipv6 route 2001:db8:4000::/48 Null0
# router bgp 64504
# neighbor 2001:db8:3000:1::1 remote-as 64503
# neighbor 2001:db8:3000:1::1 description eBGP com n3Router
# address-family ipv6

```

```
# neighbor 2001:db8:3000:1::1 activate
# network 2001:db8:4000::/48
# exit
# exit
# exit
# exit
```

O resultado dos comandos é representado pela Figura 5.19.



```
n5Router
root@n5Router:/tmp/pycore.49979/n5Router.conf# vtysh
Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n5Router# configure terminal
n5Router(config)# ipv6 route 2001:db8:4000::/48 Null0
n5Router(config)# router bgp 64504
n5Router(config-router)# neighbor 2001:db8:3000:1::1 remote-as 64503
n5Router(config-router)# neighbor 2001:db8:3000:1::1 description eBGP com n3Router
n5Router(config-router)# address-family ipv6
n5Router(config-router-af)# neighbor 2001:db8:3000:1::1 activate
n5Router(config-router-af)# network 2001:db8:4000::/48
n5Router(config-router-af)# exit
n5Router(config-router)# exit
n5Router(config)# exit
n5Router# exit
root@n5Router:/tmp/pycore.49979/n5Router.conf# █
```

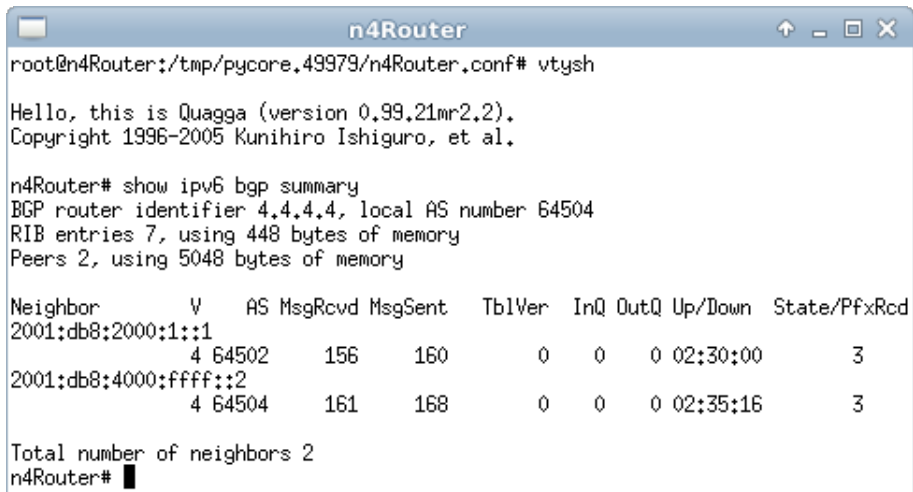
Figura 5.19: *estabelecimento da sessão eBGP entre os roteadores Quagga n5Router e n3Router.*

Os comandos são os mesmos utilizados no n4Router apenas alterando as informações relacionadas ao AS64503.

6. Verifique as configurações e o estabelecimento de sessões eBGP.
 - (a) Abra o terminal do roteador n4Router e digite os comandos:

```
# vtysh
# show ipv6 bgp summary
```

O resultado será similar ao representado pela Figura 5.20.



```

root@n4Router:~/tmp/pycore.49979/n4Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n4Router# show ipv6 bgp summary
BGP router identifier 4.4.4.4, local AS number 64504
RIB entries 7, using 448 bytes of memory
Peers 2, using 5048 bytes of memory

Neighbor      V   AS MsgRcvd MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd
2001:db8:2000:1::1
              4 64502   156    160     0    0    0 02:30:00      3
2001:db8:4000:ffff::2
              4 64504   161    168     0    0    0 02:35:16      3

Total number of neighbors 2
n4Router# █

```

Figura 5.20: exibindo a vizinhança eBGP no roteador n4Router.

É possível observar que, além do vizinho iBGP configurado anteriormente, também é listado o vizinho eBGP; e que ao final do comando são apresentadas as rotas aprendidas destes vizinhos.

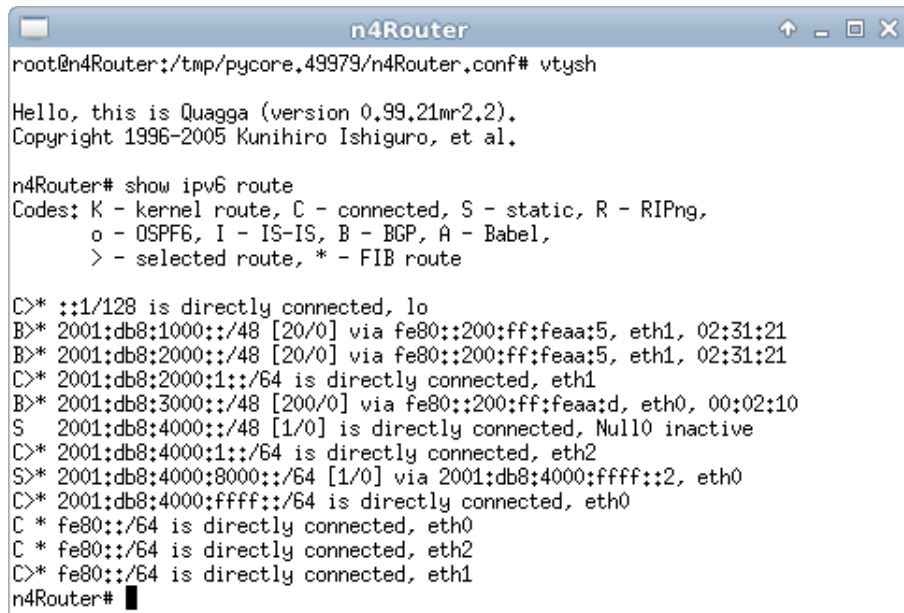
- (b) Ainda no terminal do roteador n4Router, digite o seguinte comando para verificar a tabela de rotas:

```
# vtysh
# show ipv6 route
```

O resultado será similar ao representado pela Figura 5.21.

As rotas marcadas com a letra B no início da linha são as aprendidas utilizando o protocolo BGP.

Repita os mesmos comandos no roteador n5Router e compare os resultados. Você pode utilizar também o comando `show ipv6 bgp` para ver apenas o que foi aprendido por BGP e quais os caminhos conhecidos para chegar a cada AS.



```

n4Router
root@n4Router:/tmp/pycore.49979/n4Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n4Router# show ipv6 route
Codes: K - kernel route, C - connected, S - static, R - RIPng,
       o - OSPF6, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

C>* ::1/128 is directly connected, lo
B>* 2001:db8:1000::/48 [20/0] via fe80::200:ff:feaa:5, eth1, 02:31:21
B>* 2001:db8:2000::/48 [20/0] via fe80::200:ff:feaa:5, eth1, 02:31:21
C>* 2001:db8:2000:1::/64 is directly connected, eth1
B>* 2001:db8:3000::/48 [200/0] via fe80::200:ff:feaa:d, eth0, 00:02:10
S 2001:db8:4000::/48 [1/0] is directly connected, Null0 inactive
C>* 2001:db8:4000:1::/64 is directly connected, eth2
S>* 2001:db8:4000:8000::/64 [1/0] via 2001:db8:4000:ffff::2, eth0
C>* 2001:db8:4000:ffff::/64 is directly connected, eth0
C * fe80::/64 is directly connected, eth0
C * fe80::/64 is directly connected, eth2
C>* fe80::/64 is directly connected, eth1
n4Router# █

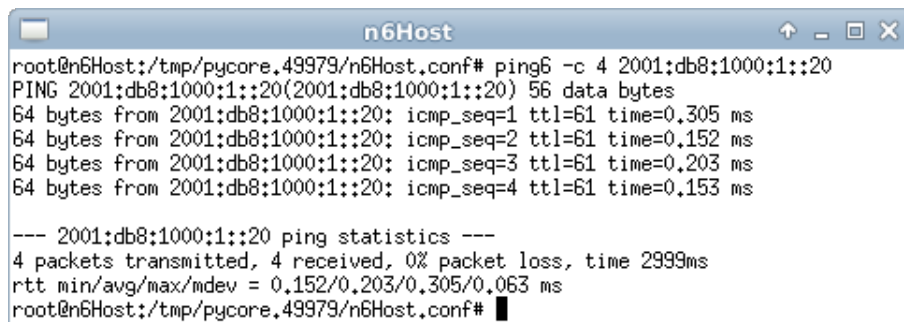
```

Figura 5.21: exibição das rotas da sessão eBGP estabelecida.

7. Por fim, repita o teste de conectividade feito no passo 3 e, a partir da máquina n6Host, execute um ping6 para a máquina n8Host.
- (a) Abra o terminal da máquina n6Host e digite o seguinte comando:

```
# ping6 -c 4 2001:db8:1000:1::20
```

O resultado do comando é representado pela Figura 5.22.



```

n6Host
root@n6Host:/tmp/pycore.49979/n6Host.conf# ping6 -c 4 2001:db8:1000:1::20
PING 2001:db8:1000:1::20(2001:db8:1000:1::20) 56 data bytes
64 bytes from 2001:db8:1000:1::20: icmp_seq=1 ttl=61 time=0,305 ms
64 bytes from 2001:db8:1000:1::20: icmp_seq=2 ttl=61 time=0,152 ms
64 bytes from 2001:db8:1000:1::20: icmp_seq=3 ttl=61 time=0,203 ms
64 bytes from 2001:db8:1000:1::20: icmp_seq=4 ttl=61 time=0,153 ms

--- 2001:db8:1000:1::20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0,152/0,203/0,305/0,063 ms
root@n6Host:/tmp/pycore.49979/n6Host.conf# █

```

Figura 5.22: novo teste de conectividade entre dois hosts de ASs diferentes.

Com este teste é possível observar que já existe conectividade entre o AS64504 e o AS64503.

Repita o teste a partir da máquina `n7Host` e veja se também há conectividade com a máquina `n8Host`. Faça os mesmos testes com o comando `traceroute6` e compare os caminhos feitos, analisando os resultados com as rotas conhecidas pelos roteadores `n4Router` e `n5Router`.

A seguir, com todas as sessões BGP estabelecidas, inicie a segunda parte do experimento aplicando políticas de roteamento de entrada e de saída para o AS64504. Até este ponto, a configuração do protocolo BGP não interfere na rota de destino a nenhuma rede, sendo que o único atributo avaliado para definir o melhor caminho é a quantidade de saltos. O primeiro passo será criar uma política de saída, AS-OUT, e influenciar o modo como o tráfego dos outros ASs chegam até o AS64504.

Essa política de saída irá tratar de acesso à Internet fatores: balanceamento do tráfego de entrada e garantir a redundância entre os dois *links* com os provedores de trânsito. Para isto, atue da seguinte forma:

- Divida o bloco /48 IPv6 em dois prefixos /49 e anuncie cada um deles por um *link*, isto é, cada /49 será conhecido por apenas um caminho. porém, se neste cenário um dos *links* ficar indisponível, o /49 anunciado por ele ficará inacessível.
- Para resolver esse problema e ter redundância entre os *links*, anuncie também o prefixo /48 pelos dois provedores. Com isso, todos os outros ASs conhecerão o /48 pelos dois caminhos.

Como os roteadores dão preferência pelo prefixo mais específico, se os dois *links* estiverem ativos, os ASs irão preferir os anúncios dos /49, ou seja, o balanceamento estará em operação. Caso um dos *links* fique indisponível, o prefixo /49 divulgado por ele deixará de ser anunciado. No entanto, como esse /49 está contido no /48, mesmo com um dos *links* desativado, os demais ASes ainda terão a opção do /48 anunciado por meio do outro provedor de trânsito, garantindo assim a redundância entre os *links*.

8. Divida o bloco `2001:db8:4000::/48` em dois prefixos /49. Identifique esses dois prefixos:

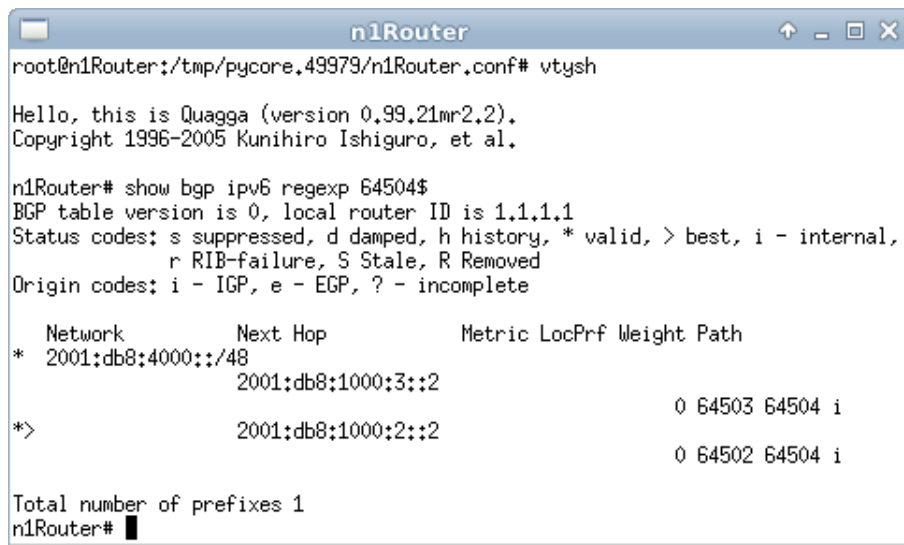
(a) _____

(b)

9. Antes de configurar os roteadores, acesse o roteador do AS64501 apenas para verificar como as rotas do AS64504 são aprendidas por ele. Acesse o roteador `n1Router` e digite os seguintes comandos:

```
# vtysh
# show bgp ipv6 regex 64504$
```

O resultado será similar ao representado pela Figura 5.23.



```
n1Router
root@n1Router:/tmp/pycore.49979/n1Router.conf# vtysh
Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n1Router# show bgp ipv6 regex 64504$
BGP table version is 0, local router ID is 1.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
               r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop          Metric LocPrf Weight Path
*  2001:db8:4000::/48
                        2001:db8:1000:3::2
                                0 64503 64504 i
*>
                        2001:db8:1000:2::2
                                0 64502 64504 i

Total number of prefixes 1
n1Router# █
```

Figura 5.23: resultado do resumo de rotas que originam do AS64504.

Este comando exibe, por meio da utilização de uma expressão regular, todas as rotas conhecidas pelo roteador `n1Router` cuja origem seja o AS64504. Como se pode observar, o prefixo `2001:db8:4000::/48` é conhecido por dois caminhos distintos, um por meio do AS64502 (vizinho do roteador `n4Router`) e outro por meio do AS64503 (vizinho do roteador `n5Router`).

10. Agora, configure inicialmente as políticas de saída no roteador `n4Router`, especificando quais prefixos serão anunciados para o AS64502.

- (a) O n4Router será configurado para divulgar o /48 e o primeiro prefixo /49. Para isto, abra o terminal do roteador n4Router e digite os seguintes comandos:

```
# vtysh
# configure terminal
# ipv6 route 2001:db8:4000::/49 Null0
# ipv6 prefix-list ANUNCIO-PARA-64502 seq 10 permit 2001:db8:4000::/48
# ipv6 prefix-list ANUNCIO-PARA-64502 seq 20 permit 2001:db8:4000::/49
# route-map BGP-OUT-64502 permit 10
# match ipv6 address prefix-list ANUNCIO-PARA-64502
# router bgp 64504
# address-family ipv6
# network 2001:db8:4000::/49
# neighbor 2001:db8:2000:1::1 route-map BGP-OUT-64502 out
# exit
# exit
# exit
# clear bgp ipv6 64502 soft out
# exit
```

O resultado dos comandos é representado pela Figura 5.24.

Estes comandos realizam as seguintes funções:

```
ipv6 route 2001:db8:4000::/49 Null0
```

Cria uma rota estática para o prefixo /49 que se pretende divulgar, apenas para forçar a sua existência na tabela de rotas.

```
ipv6 prefix-list ANUNCIO-PARA-64502 seq 10 permit 2001:db8:4000::/48
```

Cria uma prefix-list, comando utilizado para listar os prefixos que se pretende tratar. O termo ANUNCIO-PARA-64502 é apenas um nome para identificar a prefix-list. O termo seq 10 indica a ordem em que a prefix-list será executada. Caso haja mais de um prefixo na prefix-list, recomenda-se colocar valores com intervalos grandes entre si, para o caso de se adicionar futuramente novos prefixos intercalados às regras já existentes. O termo permit indica que o prefixo anunciado deve coincidir com os listados na prefix-list. Na

sequência, é indicado qual prefixo será analisado. Como pretende-se anunciar pelo roteador n4Router os prefixos 2001:db8:4000::/48 e 2001:db8:4000::/49, foi criada uma prefix-list especificando os dois prefixos.

```
route-map BGP-OUT-64502 permit 10
```

O comando route-map é utilizado para determinar qual ação será tomada de acordo com as regras especificadas. Com ele é possível descartar ou aceitar anúncios e alterar atributos. O termo BGP-OUT-64502 é apenas um nome para identificar o route-map. O termo permit indica que os prefixos verificados serão aceitos. O valor 10 indica a ordem em que o route-map será executado. Caso haja mais de uma regra no route-map, recomenda-se colocar valores com intervalos grandes entre si para o caso de se adicionar futuramente novas regras intercaladas às já existentes.

```
match ipv6 address prefix-list ANUNCIO-PARA-64502
```

É a regra a ser verificada. Neste caso, está sendo verificado se o anúncio enviado é igual ao especificado na prefix-list ANUNCIO-PARA-64502. Caso o anúncio coincida, ele será divulgado ao vizinho BGP, se não coincidir, o anúncio será automaticamente descartado.

```
neighbor 2001:db8:2000:1::1 route-map BGP-OUT-64502 out
```

Associa o route-map BGP-OUT-64502 ao vizinho 2001:db8:2000:1::1. O termo out indica que o route-map será aplicado como uma política de saída, tratando apenas os prefixos que serão anunciados ao vizinho 2001:db8:2000:1::1.

```
clear bgp ipv6 64502 soft out
```

Reenvia a tabela de roteamento ao AS64502. O protocolo BGP só envia todas as rotas conhecidas a um vizinho, quando a sessão é estabelecida. Como a sessão BGP já estava ativa, foi necessário reenviar todas as informações novamente, mas agora com a política de saída aplicada.

Os demais comandos já foram explicados em exemplos anteriores.

```

root@n4Router:/tmp/pycore.49979/n4Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n4Router# configure terminal
n4Router(config)# ipv6 route 2001:db8:4000::/49 Null0
n4Router(config)# ipv6 prefix-list ANUNCIO-PARA-64502 seq 10 permit 2001:db8:4000::/48
n4Router(config)# ipv6 prefix-list ANUNCIO-PARA-64502 seq 20 permit 2001:db8:4000::/49
n4Router(config)# route-map BGP-OUT-64502 permit 10
n4Router(config-route-map)# match ipv6 address prefix-list ANUNCIO-PARA-64502
n4Router(config-route-map)# router bgp 64504
n4Router(config-router)# address-family ipv6
n4Router(config-router-af)# network 2001:db8:4000::/49
n4Router(config-router-af)# neighbor 2001:db8:2000:1::1 route-map BGP-OUT-64502 out
n4Router(config-router-af)# exit
n4Router(config-router)# exit
n4Router(config)# exit
n4Router# clear bgp ipv6 64502 soft out
n4Router# exit
root@n4Router:/tmp/pycore.49979/n4Router.conf# █

```

Figura 5.24: *configurando o roteador n4Router.*

- (b) Para configurar as políticas de saída no roteador n5Router, abra o terminal do roteador n5Router e digite os seguintes comandos:

```

# vtysh
# configure terminal
# ipv6 route 2001:db8:4000:8000::/49 Null0
# ipv6 prefix-list ANUNCIO-PARA-64503
    seq 10 permit 2001:db8:4000::/48
# ipv6 prefix-list ANUNCIO-PARA-64503
    seq 20 permit 2001:db8:4000:8000::/49
# route-map BGP-OUT-64503 permit 10
# match ipv6 address prefix-list ANUNCIO-PARA-64503
# router bgp 64504
# address-family ipv6
# network 2001:db8:4000:8000::/49
# neighbor 2001:db8:3000:1::1 route-map BGP-OUT-64503 out
# exit
# exit
# exit

```

```
# clear bgp ipv6 64503 soft out
# exit
```

O resultado dos comandos é representado pela Figura 5.25.

```
root@n5Router:/tmp/pycore.49979/n5Router.conf# vtysh
Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n5Router# configure terminal
n5Router(config)# ipv6 route 2001:db8:4000:8000::/49 Null0
n5Router(config)# ipv6 prefix-list ANUNCIO-PARA-64503 seq 10 permit 2001:db8:4000:
0::/48
n5Router(config)# ipv6 prefix-list ANUNCIO-PARA-64503 seq 20 permit 2001:db8:4000:
0:8000::/49
n5Router(config)# route-map BGP-OUT-64503 permit 10
n5Router(config-route-map)# match ipv6 address prefix-list ANUNCIO-PARA-64503
n5Router(config-route-map)# router bgp 64504
n5Router(config-router)# address-family ipv6
n5Router(config-router-af)# network 2001:db8:4000:8000::/49
n5Router(config-router-af)# neighbor 2001:db8:3000:1::1 route-map BGP-OUT-64503
out
n5Router(config-router-af)# exit
n5Router(config-router)# exit
n5Router(config)# exit
n5Router# clear bgp ipv6 64503 soft out
n5Router# exit
root@n5Router:/tmp/pycore.49979/n5Router.conf# █
```

Figura 5.25: configurando o roteador n5Router.

Os comandos são os mesmos utilizados no n4Router, com apenas as informações relacionadas ao AS64503 alteradas.

11. Para verificar se os anúncios foram feitos corretamente, acesse o roteador n1Router novamente e visualize as informações de sua tabela de rotas BGP. Para isto, abra o terminal do roteador n1Router e digite o seguinte comando:

```
# vtysh
# show bgp ipv6 regexp 64504$
```

O resultado destes comandos será similar ao representado pela Figura 5.26.

```

n1Router
root@n1Router:/tmp/pycore.49979/n1Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n1Router# show bgp ipv6 regexp 64504$
BGP table version is 0, local router ID is 1.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
               r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop          Metric LocPrf Weight Path
* 2001:db8:4000::/48
                   2001:db8:1000:3::2
                   0 64503 64504 i
*>
                   2001:db8:1000:2::2
                   0 64502 64504 i
* 2001:db8:4000::/49
                   2001:db8:1000:3::2
                   0 64503 64502 64504 i
*>
                   2001:db8:1000:2::2
                   0 64502 64504 i
*> 2001:db8:4000:8000::/49
                   2001:db8:1000:3::2
                   0 64503 64504 i
*
                   2001:db8:1000:2::2
                   0 64502 64503 64504 i

Total number of prefixes 3
n1Router# █

```

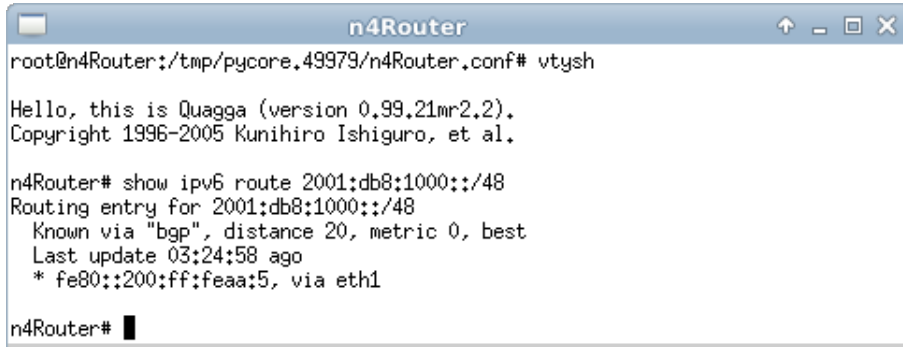
Figura 5.26: resumo de rotas bgp do roteador n1Router.

É possível observar que os dois prefixos /49 estão sendo divulgados, cada um por um caminho distinto. O primeiro /49 é divulgado pelo AS64502 e o segundo bloco /49 é divulgado apenas pelo *link* conectado ao AS64503. O prefixo /48 é divulgado pelos dois *links* simultaneamente.

12. Feita a política de saída, o próximo passo é criar uma política de roteamento de entrada. As políticas de entrada são aplicadas sobre os anúncios aprendidos dos outros ASs e, teoricamente, é possível criar uma política e influenciar individualmente a tomada de decisão de roteamento para cada prefixo da Internet. Neste exercício, a política de entrada será aplicada aos prefixos anunciados pelo AS64501.
 - (a) Inicialmente, verifique como o AS64504 aprendeu os caminhos até as redes do AS64501. Abra o terminal do roteador n4Router e digite o seguinte comando:

```
# vtysh
# show ipv6 route 2001:db8:1000::/48
```

O resultado dos comandos é representado pela Figura 5.27.



```
n4Router
root@n4Router:/tmp/pycore.49979/n4Router.conf# vtysh
Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n4Router# show ipv6 route 2001:db8:1000::/48
Routing entry for 2001:db8:1000::/48
  Known via "bgp", distance 20, metric 0, best
  Last update 03:24:58 ago
  * fe80::200:ff:feaa:5, via eth1

n4Router# █
```

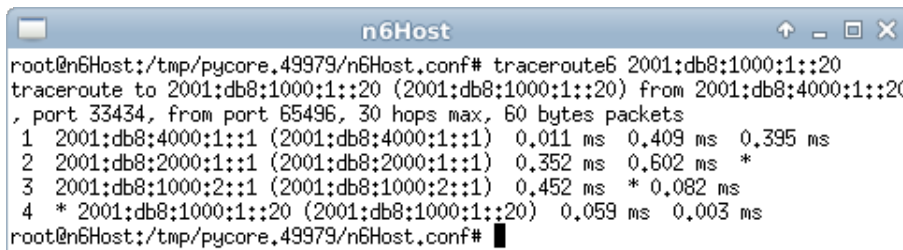
Figura 5.27: informações sobre a rede 2001:db8:1000::/48.

Este comando mostra as informações existentes na tabela de roteamento sobre a rede 2001:db8:1000::/48. É possível observar que o caminho para chegar até ela é através do AS64502, utilizando a interface eth1.

- (b) Outra forma de verificar o caminho até o AS64501 é utilizando o comando `traceroute6`. Abra o terminal da máquina `n6Host` e execute o seguinte comando:

```
# traceroute6 2001:db8:1000:1::20
```

O resultado do comando é representado pela Figura 5.28.



```
n6Host
root@n6Host:/tmp/pycore.49979/n6Host.conf# traceroute6 2001:db8:1000:1::20
traceroute to 2001:db8:1000:1::20 (2001:db8:1000:1::20) from 2001:db8:4000:1::20
, port 33434, from port 65496, 30 hops max, 60 bytes packets
 1 2001:db8:4000:1::1 (2001:db8:4000:1::1) 0,011 ms 0,409 ms 0,395 ms
 2 2001:db8:2000:1::1 (2001:db8:2000:1::1) 0,352 ms 0,602 ms *
 3 2001:db8:1000:2::1 (2001:db8:1000:2::1) 0,452 ms * 0,082 ms
 4 * 2001:db8:1000:1::20 (2001:db8:1000:1::20) 0,059 ms 0,003 ms

root@n6Host:/tmp/pycore.49979/n6Host.conf# █
```

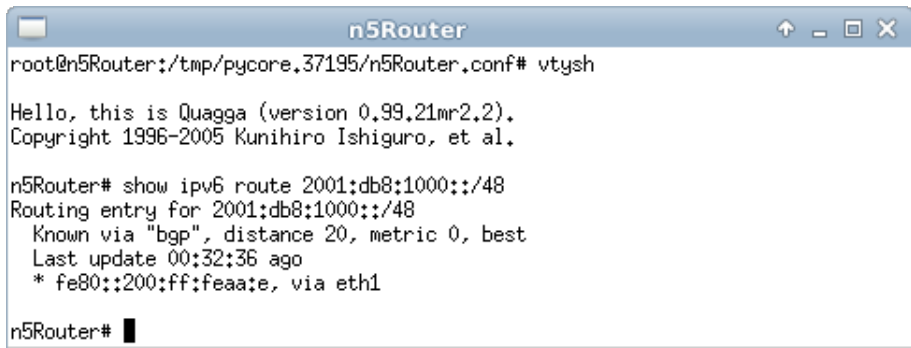
Figura 5.28: informações da rota percorrida pelos pacotes.

Este comando traça o caminho do `n6Host` até o `n8Host`, localizado no AS64501. É possível observar que caminho até ele passa pelos roteadores `n4Router`, `n3Router` e `n1Router`, até chegar ao destino. Este caminho foi o escolhido por ser o caminho mais curto.

- (c) Realize os mesmos testes a partir do roteador n5Router e da máquina n7Host. Abra o terminal do roteador n5Router e digite o seguinte comando:

```
# vtysh
# show ipv6 route 2001:db8:1000::/48
```

O resultado dos comandos é representado pela Figura 5.29.



```
n5Router
root@n5Router:/tmp/pycore.37195/n5Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n5Router# show ipv6 route 2001:db8:1000::/48
Routing entry for 2001:db8:1000::/48
  Known via "bgp", distance 20, metric 0, best
  Last update 00:32:36 ago
  * fe80::200:ff:feaa:e, via eth1

n5Router# █
```

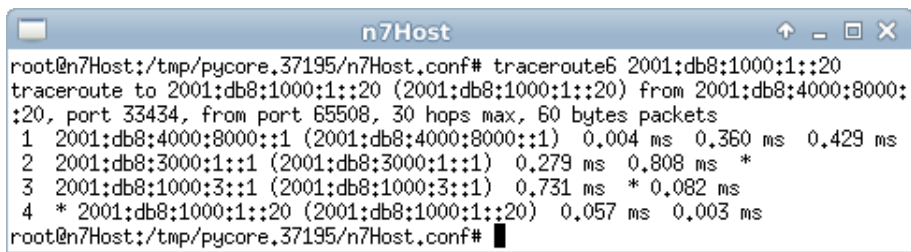
Figura 5.29: informações das rotas sobre a rede 2001:db8:1000::/48.

Assim como no teste feito no roteador n4Router, este comando mostra as informações existentes na tabela de roteamento sobre a rede 2001:db8:1000::/48. É possível observar que, a partir do roteador n5Router, o caminho para chegar até ela é através do AS64503, utilizando a interface eth1.

- (d) Faça o teste com o comando traceroute6 a partir da máquina n7Host. Para tanto, abra o terminal da máquina n7Host e execute o seguinte comando:

```
# traceroute6 2001:db8:1000:1::20
```

O resultado do comando é representado pela Figura 5.30.



```
n7Host
root@n7Host:/tmp/pycore.37195/n7Host.conf# traceroute6 2001:db8:1000:1::20
traceroute to 2001:db8:1000:1::20 (2001:db8:1000:1::20) from 2001:db8:4000:8000:
:20, port 33434, from port 65508, 30 hops max, 60 bytes packets
 1 2001:db8:4000:8000::1 (2001:db8:4000:8000::1) 0.004 ms 0.360 ms 0.429 ms
 2 2001:db8:3000:1::1 (2001:db8:3000:1::1) 0.279 ms 0.808 ms *
 3 2001:db8:1000:3::1 (2001:db8:1000:3::1) 0.731 ms * 0.082 ms
 4 * 2001:db8:1000:1::20 (2001:db8:1000:1::20) 0.057 ms 0.003 ms

root@n7Host:/tmp/pycore.37195/n7Host.conf# █
```

Figura 5.30: informações da rota percorrida pelos pacotes.

Como visto anteriormente, este comando traça o caminho do n7Host até o n8Host. É possível observar que, neste caso, o caminho até o n8Host passa pelos roteadores n5Router, n3Router e n1Router até chegar ao destino. Este caminho foi o escolhido por ser o mais curto.

- (e) Configure a política de entrada de modo que o tráfego da rede até o AS64501 passe exclusivamente pelo roteador n4Router, aumentando a preferência por este caminho, independente deste ser o melhor ou o pior caminho. Para tanto, abra o terminal do roteador n4Router e execute os seguintes comandos:

```
# vtysh
# configure terminal
# ipv6 prefix-list PREFIX0-D0-64501
  seq 10 permit 2001:db8:1000::/48
# ip as-path access-list ORIGEM-N0-AS64501 permit 64501$
# route-map BGP-IN-64502 permit 10
# match as-path ORIGEM-N0-AS64501
# match ipv6 address prefix-list PREFIX0-D0-64501
# set local-preference 150
# router bgp 64504
# address-family ipv6
# neighbor 2001:db8:2000:1::1 route-map BGP-IN-64502 in
# exit
# exit
# exit
# clear bgp ipv6 64502 soft in
# exit
```

O resultado dos comandos é representado pela Figura 5.31.

```

root@n4Router:/tmp/pycore.49979/n4Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n4Router# configure terminal
n4Router(config)# ipv6 prefix-list PREFIX0-D0-64501 seq 10 permit 2001:db8:1000::/48
n4Router(config)# ip as-path access-list ORIGEM-N0-AS64501 permit 64501$
n4Router(config)# route-map BGP-IN-64502 permit 10
n4Router(config-route-map)# match as-path ORIGEM-N0-AS64501
n4Router(config-route-map)# match ipv6 address prefix-list PREFIX0-D0-64501
n4Router(config-route-map)# set local-preference 150
n4Router(config-route-map)# router bgp 64504
n4Router(config-router)# address-family ipv6
n4Router(config-router-af)# neighbor 2001:db8:2000:1::1 route-map BGP-IN-64502 in
n4Router(config-router-af)# exit
n4Router(config-router)# exit
n4Router(config)# exit
n4Router# clear bgp ipv6 64502 soft in
n4Router# exit
root@n4Router:/tmp/pycore.49979/n4Router.conf# █

```

Figura 5.31: configurações das políticas de entrada do roteador n4Router.

Estes comandos verificam se o anúncio recebido é do prefixo 2001:db8:1000::/48 e se a origem deste anúncio foi o AS64501. Caso estas duas condições sejam preenchidas, aumenta-se o valor do atributo local-preference. Os comandos possuem as seguintes funções:

```
ipv6 prefix-list PREFIX0-D0-64501 seq 10 permit 2001:db8:1000::/48
```

Nesta prefix-list indica-se que será analisado apenas o anúncio do prefixo 2001:db8:1000::/48.

```
ip as-path access-list ORIGEM-N0-AS64501 permit 64501$
```

Este comando cria, por meio de expressão regular, regras que analisam o as-path, ou seja, analisam o caminho para chegar a uma determinada rede. Neste exemplo, a expressão regular é validada se a origem do anúncio é o AS64501.

```
route-map BGP-IN-64502 permit 10
```

O comando route-map é utilizado para determinar qual ação será tomada, de acordo com as regras especificadas. Neste caso, se o anúncio recebido coincidir com as duas regras, ele será aceito e o valor do atributo local-preference será alterado para 150.

```
match as-path ORIGEM-N0-AS64501
```

A primeira regra a ser verificada é se o anúncio recebido tem origem no AS64501, como definido no as-path ORIGEM-N0-AS64501.

```
match ipv6 address prefix-list PREFIX0-D0-64501
```

A segunda regra analisada verifica se o anúncio recebido é o do prefixo 2001:db8:1000::/48, como definido na prefix-list PREFIX0-D0-64501.

```
set local-preference 150
```

Altera o valor do atributo local-preference para 150, caso as condições especificadas sejam validadas. O valor padrão do atributo local-preference é igual a 100 e, quanto maior o valor, maior a preferência.

```
neighbor 2001:db8:2000:1::1 route-map BGP-IN-64502 in
```

Associa o route-map BGP-IN-64502 ao vizinho 2001:db8:2000:1::1. O termo in indica que o route-map será aplicado como uma política de entrada, tratando apenas os prefixos recebidos pelo vizinho 2001:db8:2000:1::1.

```
clear bgp ipv6 64502 soft in
```

Solicita o reenvio da tabela de roteamento ao AS64502. O protocolo BGP só envia todas as rotas conhecidas a um vizinho, quando a sessão é estabelecida. Como a sessão BGP já estava ativa, foi necessário solicitar o reenvio de todas as informações novamente para possibilitar a aplicação da nova política de entrada.

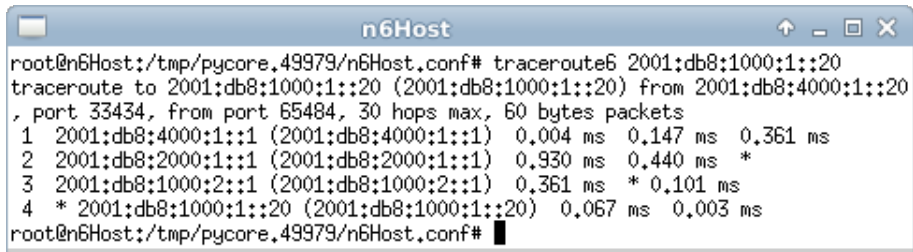
Os demais comandos já foram explicados anteriormente.

13. Repita os testes com o comando `traceroute6` para verificar se houve alguma mudança no caminho até o AS64501.

(a) Abra o terminal da máquina `n6Host` e execute o seguinte comando:

```
# traceroute6 2001:db8:1000:1::20
```

O resultado do comando é representado pela Figura 5.32.



```
n6Host
root@n6Host:/tmp/pycore.49979/n6Host.conf# traceroute6 2001:db8:1000:1::20
traceroute to 2001:db8:1000:1::20 (2001:db8:1000:1::20) from 2001:db8:4000:1::20
, port 33434, from port 65484, 30 hops max, 60 bytes packets
 1 2001:db8:4000:1::1 (2001:db8:4000:1::1)  0,004 ms  0,147 ms  0,361 ms
 2 2001:db8:2000:1::1 (2001:db8:2000:1::1)  0,930 ms  0,440 ms  *
 3 2001:db8:1000:2::1 (2001:db8:1000:2::1)  0,361 ms  *  0,101 ms
 4 * 2001:db8:1000:1::20 (2001:db8:1000:1::20)  0,067 ms  0,003 ms
root@n6Host:/tmp/pycore.49979/n6Host.conf# █
```

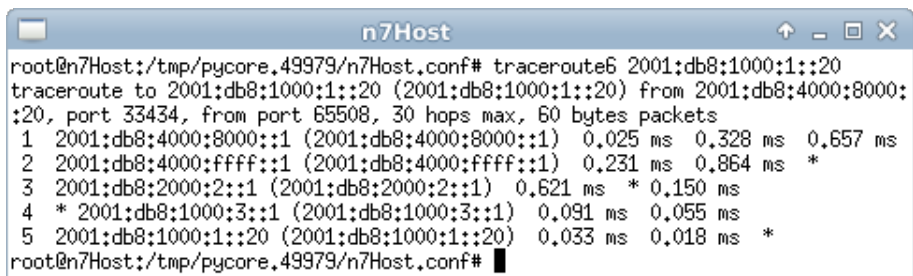
Figura 5.32: informações da rota percorrida pelos pacotes.

Como foi aumentada a preferência para que o tráfego com destino às redes do AS64501 saísse pelo roteador `n4Router` e este já era o melhor caminho para estas redes, não houve alteração no caminho do `n6Host` até o `n8Host`.

(b) Agora, repita o teste a partir do `n7Host`. Abra o terminal da máquina `n7Host` e execute o seguinte comando:

```
# traceroute6 2001:db8:1000:1::20
```

O resultado do comando é representado pela Figura 5.33.



```
n7Host
root@n7Host:/tmp/pycore.49979/n7Host.conf# traceroute6 2001:db8:1000:1::20
traceroute to 2001:db8:1000:1::20 (2001:db8:1000:1::20) from 2001:db8:4000:8000:
:20, port 33434, from port 65508, 30 hops max, 60 bytes packets
 1 2001:db8:4000:8000::1 (2001:db8:4000:8000::1)  0,025 ms  0,328 ms  0,657 ms
 2 2001:db8:4000:ffff::1 (2001:db8:4000:ffff::1)  0,231 ms  0,864 ms  *
 3 2001:db8:2000:2::1 (2001:db8:2000:2::1)  0,621 ms  *  0,150 ms
 4 * 2001:db8:1000:3::1 (2001:db8:1000:3::1)  0,091 ms  0,055 ms
 5 2001:db8:1000:1::20 (2001:db8:1000:1::20)  0,033 ms  0,018 ms  *
root@n7Host:/tmp/pycore.49979/n7Host.conf# █
```

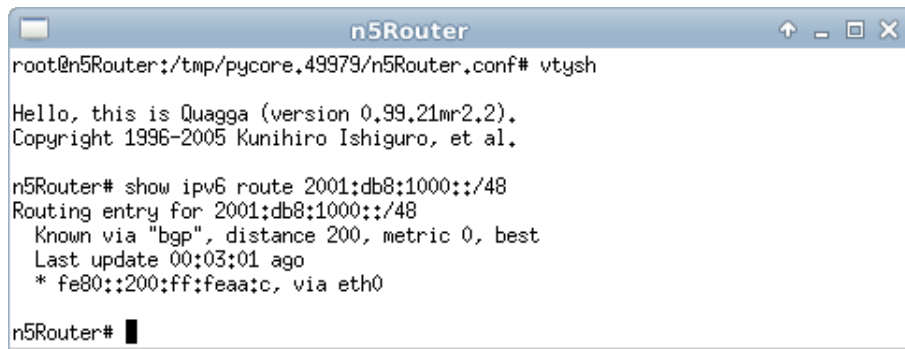
Figura 5.33: informações da rota percorrida pelos pacotes.

Neste caso, comparando com o teste realizado anteriormente, é possível notar que agora, para chegar ao n8Host, o n7Host dá-se um salto a mais, passando pelo roteador n4Router. Isto ocorre porque foi aumentada a preferência do caminho para o AS64501 através do roteador n4Router, aumentando o valor do atributo `local-preference`.

- (c) Acesse o roteador n5Router para verificar se houve alguma alteração em sua tabela de roteamento. Para tanto, abra o terminal do n5Router e execute o seguinte comando:

```
# vtysh
# show ipv6 route 2001:db8:1000::/48
```

O resultado dos comandos é representado pela Figura 5.34.



```
n5Router
root@n5Router:/tmp/pycore.49979/n5Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n5Router# show ipv6 route 2001:db8:1000::/48
Routing entry for 2001:db8:1000::/48
  Known via "bgp", distance 200, metric 0, best
  Last update 00:03:01 ago
  * fe80::200:ff:feaa:c, via eth0

n5Router# █
```

Figura 5.34: informações das rotas do roteador n5Router.

Veja que agora o caminho escolhido para chegar às redes do AS64501 é passa por n4Router, utilizando a interface eth0, mesmo ele sendo o caminho mais longo. Isto ocorre porque o atributo `local-preference` tem um peso maior na escolha do caminho em relação à distância.

- (d) Verifique também as informações da tabela BGP do roteador n5Router. Ainda no terminal do n5Router, execute o seguinte comando:

```
# vtysh
# show bgp ipv6 2001:db8:1000::/48
```

O resultado dos comandos é representado pela Figura 5.35.

```

root@n5Router:/tmp/pycore.49979/n5Router.conf# vtysh

Hello, this is Quagga (version 0.99.21mr2.2).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

n5Router# show bgp ipv6 2001:db8:1000::/48
BGP routing table entry for 2001:db8:1000::/48
Paths: (2 available, best #2, table Default-IP-Routing-Table)
  Not advertised to any peer
  64503 64501
    2001:db8:3000:1::1 from 2001:db8:3000:1::1 (3.3.3.3)
    (fe80::200:ff:feaa:e)
    Origin IGP, localpref 100, valid, external
    Last update: Wed Dec 18 14:24:15 2013

  64502 64501
    2001:db8:4000:ffff::1 from 2001:db8:4000:ffff::1 (4.4.4.4)
    (fe80::200:ff:feaa:c)
    Origin IGP, localpref 150, valid, internal, best
    Last update: Wed Dec 18 15:29:42 2013

n5Router# █

```

Figura 5.35: informações das rotas BGP do roteador n5Router.

É possível verificar que o roteador n5Router conhece dois caminhos para chegar ao AS64501. Em um deles o valor do atributo local-preference é o valor padrão 100. Na outra opção, o valor do local-preference é 150 tornando essa rota preferencial. Lembre que este valor foi alterado no roteador n4Router e repassado por iBGP ao roteador n5Router.

A aplicação de políticas de entrada pode ser utilizada em diversas situações. Neste exercício simplesmente optou-se por enviar os pacotes com destino ao AS64501 por um único caminho. Esta escolha poderia ter sido justificada pelo fato de um dos *links* ter maior capacidade, por ser mais barato, por um ser o *link* principal e o outro ser um *backup*, etc.

14. Encerre a simulação, conforme descrito no Apêndice B.

Apêndice A

Instalação de pacotes quando não é utilizada a VM do IPv6.br

Para a elaboração dos laboratórios apresentados neste texto, foi preparada uma máquina virtual (*Virtual Machine* – VM) utilizando o VirtualBox como ferramenta de virtualização (Oracle, 2015), Linux como sistema operacional e Xubuntu 12.04 LTS (Precise Pangolin) 32 *bits* como distribuição Linux (Canonical, 2012).

Por meio do repositório oficial do Ubuntu, foram instalados os seguintes pacotes:

CORE

- libev4
- libtk-img
- quagga-mr (Naval Research Laboratory, 2012)
- core (Naval Research Laboratory, 2013)

Editores de texto

- emacs
- nano
- vim-gtk

Ferramentas de rede

- bridge-utils
- ebtables
- mtr
- iputils-ping
- traceroute
- whois
- wireshark

Ferramentas específicas de IPv6

- ndisc6
- radvd
- rdnssd

Serviços

- apache2
- bind9
- cifs-utils
- nginx
- openssh-server
- samba
- squid3

Geração de Live DVD/USB

- remastersys-gui (Brijeski, 2012)

Alguns pacotes necessitam que seja compilado seu código-fonte para instalação, visto que algumas funcionalidades necessárias não estavam disponíveis na versão mantida do pacote.

DHCP 4.2.4-P2

```
$ wget ftp://ftp.isc.org/isc/dhcp/4.2.4-P2/dhcp-4.2.4-P2.tar.gz
$ tar xvzf dhcp-4.2.4-P2.tar.gz
$ cd dhcp-4.2.4-P2/
$ ./configure
$ make
$ sudo make install
```

Dibbler 0.8.4RC1

```
$ wget http://klub.com.pl/dhcpv6/dibbler/dibbler-0.8.4RC1-
  src.tar.gz
$ tar xvzf dibbler-0.8.4RC1-src.tar.gz
$ cd dibbler-0.8.4RC1/
$ ./configure
$ make
$ sudo make install
```

THC-IPv6

```
$ wget http://www.thc.org/releases/thc-ipv6-2.3.tar.gz
$ tar xvzf thc-ipv6-2.3.tar.gz
$ cd thc-ipv6-2.3
$ make
$ sudo make install
```

NDPMon

```
$ wget http://downloads.sourceforge.net/project/ndpmon/ndpmon/  
ndpmon-2.1/ndpmon_2.1.0.tar.gz  
$ tar xvzf ndpmon_2.1.0.tar.gz  
$ cd ndpmon_2.1.0  
$ sudo apt-get install build-essential libpcap-dev libssl-dev  
libtool autoconf automake autotools-dev libxml2-dev  
libxslt1-dev bsd-mailx wireshark  
$ sudo autoconf  
$ ./configure  
$ make  
$ sudo make install
```

Apêndice B

Emulador de redes CORE

O que é o CORE?

Desenvolvido pela divisão de pesquisa e tecnologia da *Boeing*, o *Common Open Research Emulator* – CORE (Naval Research Laboratory, 2009) é uma ferramenta utilizada para emular redes de computadores. Por meio dele é possível simular topologias de rede contendo máquinas Unix e equipamentos de redes como roteadores e *switches*.

Instalando o CORE

Para instalar o CORE pode-se baixar o programa direto da página *Web* do próprio CORE (<http://www.nrl.navy.mil/itd/ncs/products/core>). Também é possível realizar a instalação à partir do código-fonte disponível em <http://code.google.com/p/coreemu/>.

Operações básicas

A seguir serão demonstradas como realizar as operações mais básicas dentro do emulador CORE.

1. **Iniciando o CORE**

Para iniciar o CORE, pode-se utilizar o terminal ou executá-lo diretamente no **Desktop**. Para iniciá-lo utilizando o terminal, basta executar a seguinte instrução:

```
# core-gui
```

Isto fará com que a interface gráfica do CORE seja aberta. Também é possível fazer isto por meio da própria interface gráfica, utilizando um duplo-clique no ícone do CORE, como mostra na Figura B.1.



Figura B.1: *abrindo o CORE utilizando interface gráfica.*

2. Abrindo uma experiência

Para abrir uma experiência, utilize um duplo-clique diretamente no arquivo contendo a topologia (**arquivo .imn**), como mostra a Figura B.2.

Também é possível abrir a experiência a partir do próprio CORE, utilizando a opção localizada em File->Open e selecionando o arquivo imn desejado, como representado na Figura B.3.

É importante destacar que ao abrir uma nova experiência, deve-se encerrar a experiência que estiver aberta. Isto é necessário, pois podem ocorrer problemas caso sejam executadas duas experiências simultaneamente.

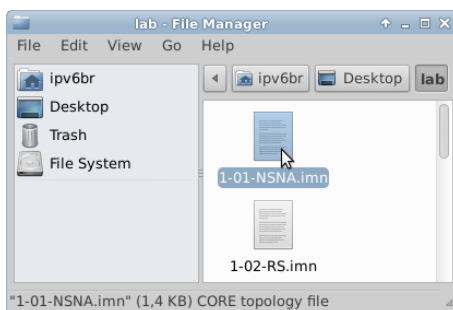


Figura B.2: abrindo uma experiência diretamente por meio de um arquivo *imn*.

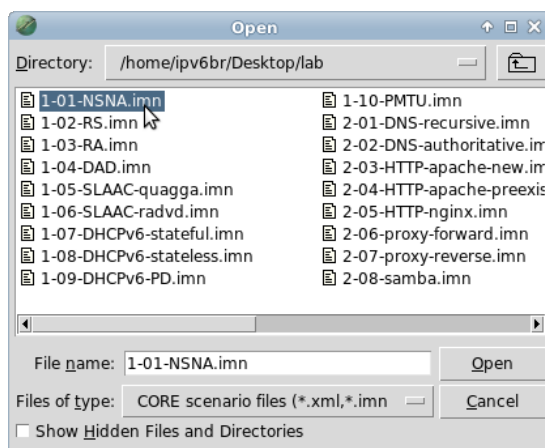


Figura B.3: abrindo uma experiência por meio do CORE.

3. Iniciando uma experiência

Para iniciar uma experiência, basta clicar no botão de *Start the Session*, como mostra a Figura B.4.

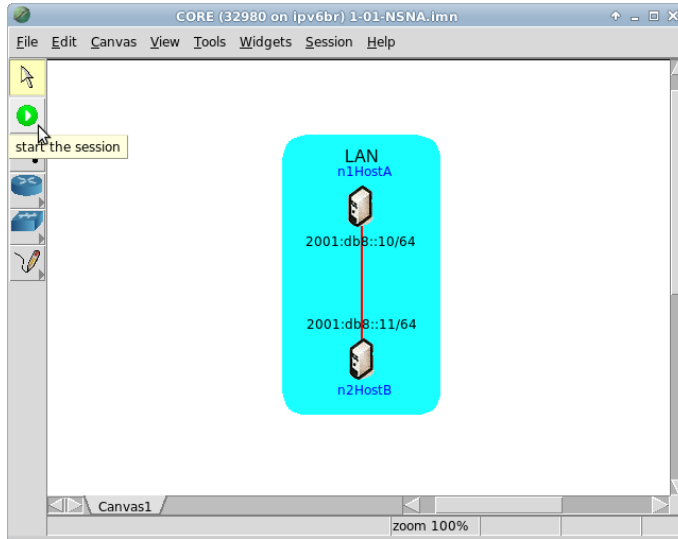


Figura B.4: *iniciando uma experiência do CORE.*

Para que o CORE funcione corretamente é necessário esperar que toda a experiência carregue por completo. Ao iniciar a experiência, é possível notar que quadrados coloridos se formam em torno dos equipamentos da topologia. É necessário esperar que estes quadrados desapareçam por completo para iniciar a experiência, como mostra a Figura B.5 e a Figura B.6.

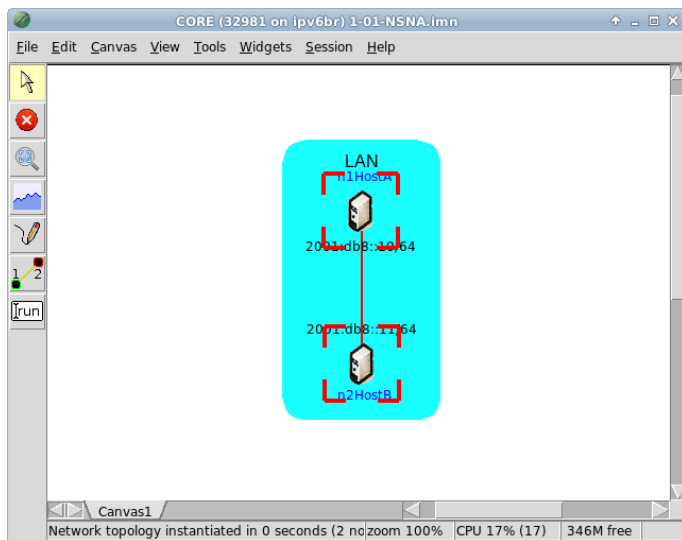


Figura B.5: carregamento dos equipamentos da topologia do CORE.

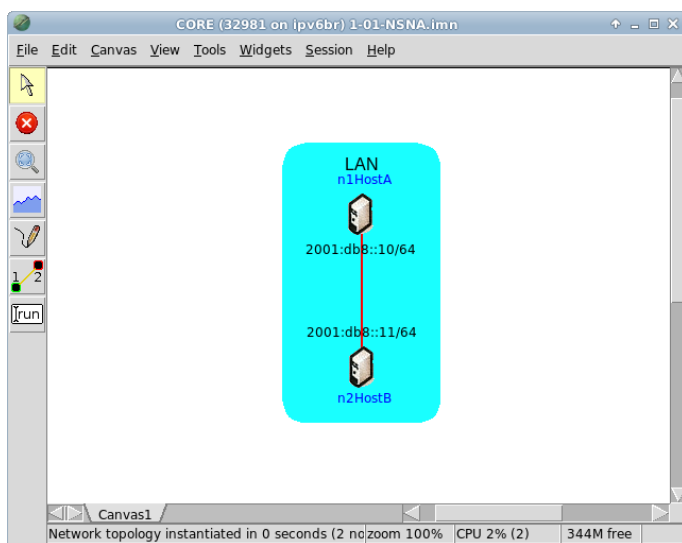


Figura B.6: finalização do carregamento.

4. Encerrando uma experiência

Para encerrar uma experiência, basta clicar no botão de *Stop the Session*, como mostra a Figura B.7.

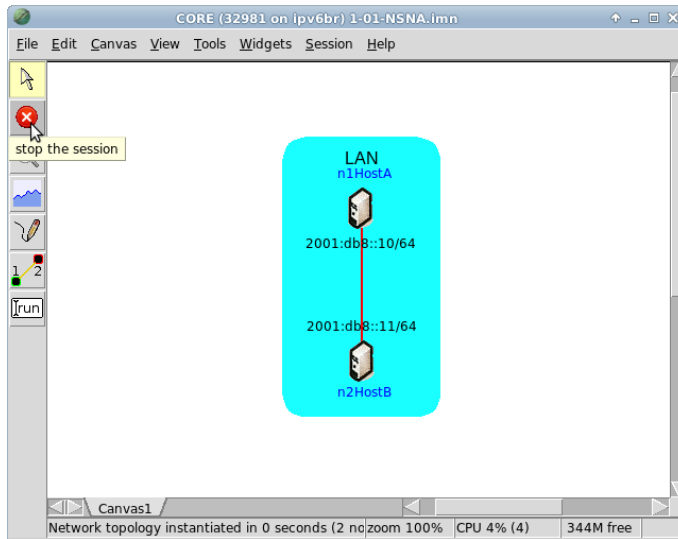


Figura B.7: encerrando uma experiência do CORE.

Novamente é necessário esperar que os quadrados que se formam em volta dos equipamentos da topologia desapareçam por completo. Isto é necessário, pois o CORE utiliza recursos da própria máquina em que está instalado para criar instâncias desses equipamentos. Por isso, é preciso aguardar o encerramento total dos processos para evitar que algum processo continue sendo executado mesmo após o encerramento do CORE.

5. Editando uma topologia

Primeiramente, certifique-se de que não há nenhuma experiência sendo executada. Isto é necessário, pois o modo de edição só funciona caso o experimento não esteja em execução. Dentro do CORE existem diversas funcionalidades para serem utilizadas em sua topologia.

- (a) Para adicionar nós à topologia, basta selecionar o equipamento desejado (*router*, *hub*, *PC*, *host*) e em seguida clicar na posição desejada para o equipamento, como apresentado na Figura B.8 e na Figura B.9.

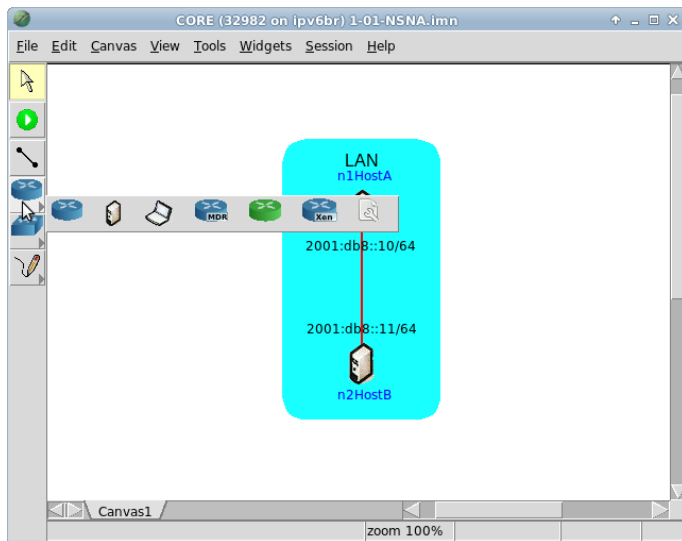


Figura B.8: selecionando um equipamento para adicionar à topologia.

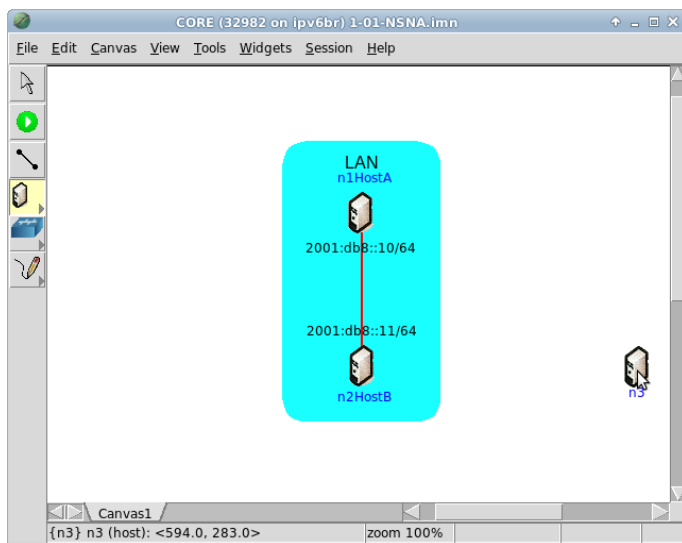


Figura B.9: adicionando o equipamento à topologia.

- (b) Para interconectar os nós da topologia, basta selecionar a ferramenta *link tool*, clicar em um dos nós e, segurando o botão do mouse, arrastá-lo até o outro nó, como demonstrado na Figura B.10 e na Figura B.11.

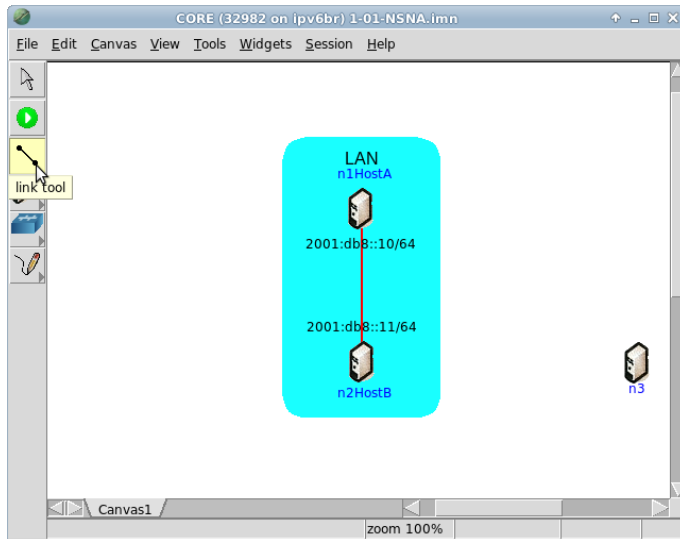


Figura B.10: selecionando a ferramenta *link tool*.

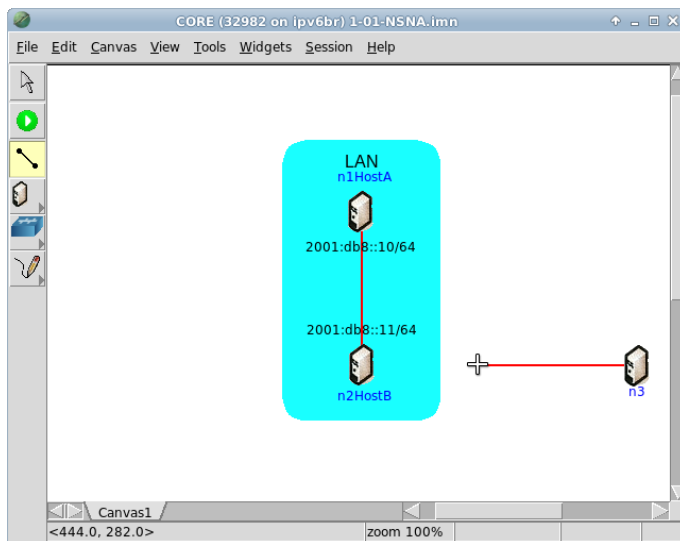


Figura B.11: interconexão dos nós feita.

- (c) Para configurar os serviços executados em determinado nó, realize um duplo-clique sobre ele. Como pode-se observar na Figura B.12, um menu de configurações será aberto.

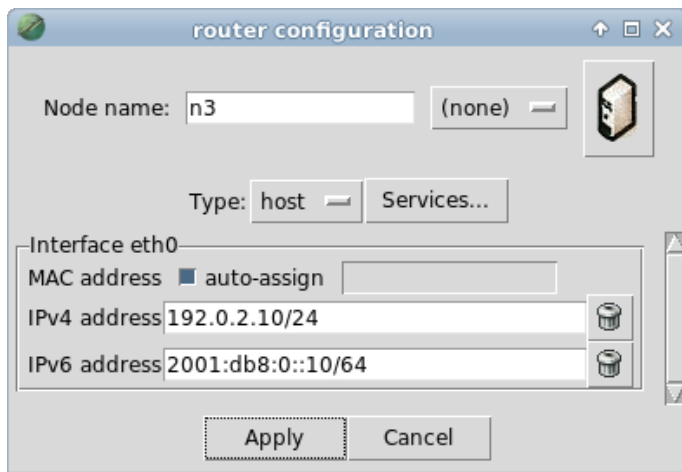


Figura B.12: menu de configurações de um equipamento.

Neste menu é possível configurar diversas características do nó, como endereço IPv4 e IPv6, endereço MAC além de diversos serviços que podem ser habilitados ou personalizados. Para configurar os serviços que serão utilizados no equipamento, clique em *Services...*, do modo que está representado na Figura B.13.

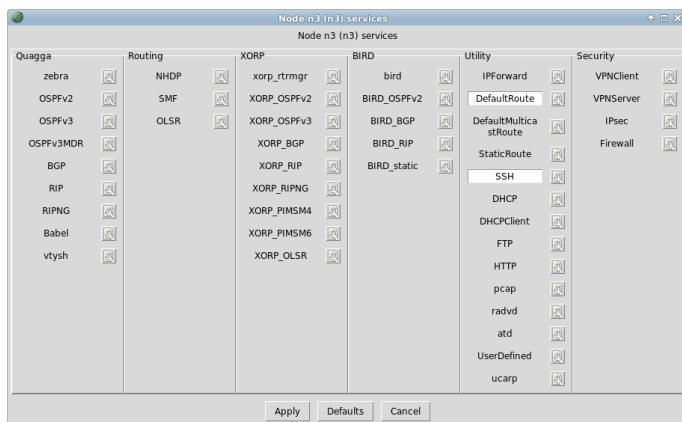


Figura B.13: menu de configurações dos serviços de um equipamento.

Os ícones selecionados são os que serão executados ao inicializar o experimento. Também é possível personalizar o que cada *script* fará durante sua execução. Para isto basta clicar no ícone ao lado do serviço desejado, como demonstrado na Figura B.14.

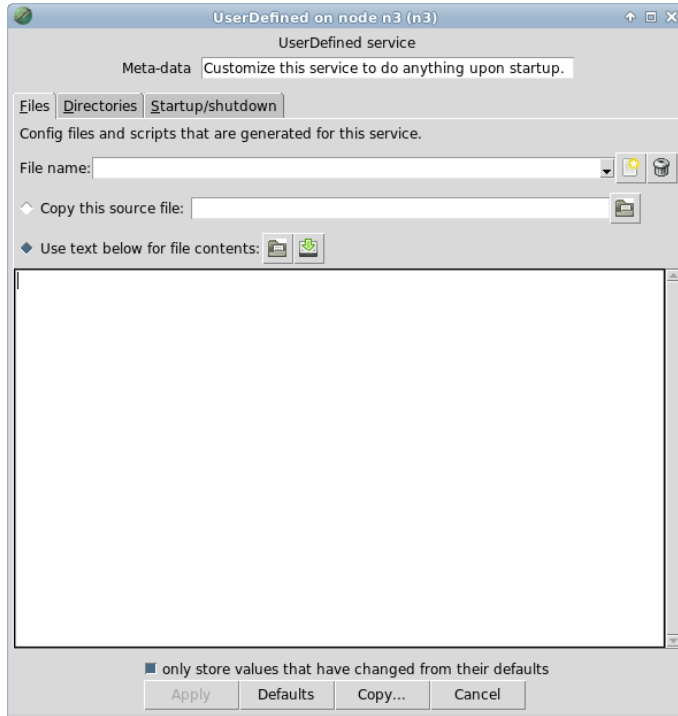


Figura B.14: *menu de personalização de um serviço.*

- (d) O CORE permite o uso de *scripts* antes e depois da execução do experimento. Para isso utiliza-se a ferramenta *Hooks*, localizada em *Session->Hooks*. A Figura B.15 ilustra esta situação.

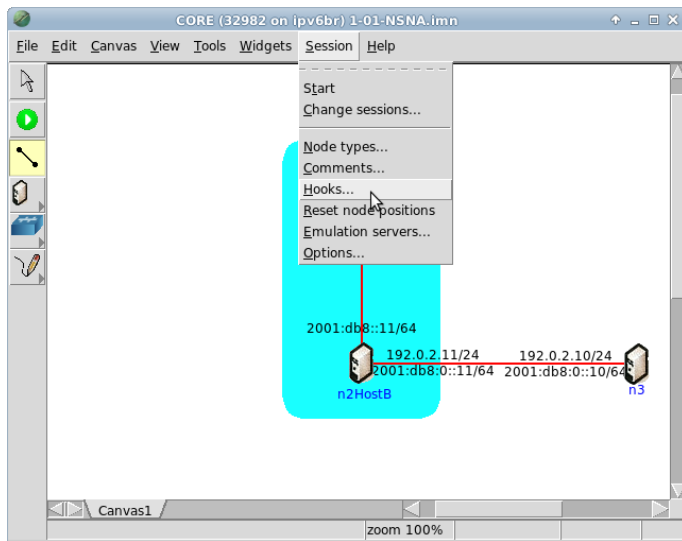


Figura B.15: abrindo o menu de hooks do experimento.

Apêndice C

Comandos básicos

Como abrir um terminal da VM fora do CORE

Em algumas experiências é necessário que se abra o terminal da própria máquina virtual, ou seja, um terminal fora do emulador CORE. A forma mais simples é clicar no ícone do terminal disponível na barra de menus superior e no próprio Desktop, conforme ilustrado na Figura C.1.

Acessando a VM como root

Em algumas experiências pode ser necessário acessar a máquina virtual como root. O procedimento para tal é bastante simples, basta abrir um terminal da VM fora do CORE e utilizar o seguinte comando:

```
# sudo su
```

A senha para o acesso como root é: `ipv6br`.



Figura C.1: atalhos para abrir um terminal.

Como verificar endereços de interfaces em nós

Uma forma de verificar os endereços de uma interface é por meio do comando `ifconfig`. O `ifconfig` faz parte de uma coleção de ferramentas chamada `net-tools`. Porém, esta ferramenta já está obsoleta e sua última atualização ocorreu em 2001.

Para substituí-la, foi criada uma nova coleção de ferramentas denominada `iproute2`. O `iproute2` substituiu completamente a ferramenta antiga e possui um suporte melhor para o IPv6 que o `ifconfig`. Por isto deve-se utilizar o comando `ip` no lugar do antigo `ifconfig`.

Para verificar os endereços de todas as interfaces, deve-se utilizar o seguinte comando:

```
# ip addr show
```

Caso seja necessário verificar apenas o endereço de uma única interface conhecida, pode-se utilizar o comando:

```
# ip addr show dev <nome-da-interface>
```

Para identificar o endereço IP da interface, basta procurar pelos campos de nome `inet` para endereços IPv4 e `inet6` para endereços IPv6.

Teste de conectividade

IPv4

Para testar a conectividade entre dois nós em IPv4, pode-se utilizar o comando `ping` em um dos nós, com o outro nó como destino:

```
# ping <endereço IPv4 de destino>
```

IPv6

Para testar a conectividade entre dois nós em IPv6, pode-se utilizar o comando `ping6` em um dos nós, com o outro nó como destino:

```
# ping6 <endereço IPv6 de destino>
```

Caso o endereço de destino seja um endereço IPv6 do tipo *link-local*, deve-se especificar a interface de saída da mensagem com o seguinte comando:

```
# ping6 -I <nome da interface> <endereço IPv6 link-local de destino>
```

Analisando o resultado

Com o comando ping, é verificado se existe ou não conectividade entre os nós da rede. Ao se realizar um ping, é enviado um pacote ICMP do tipo *echo request* ao endereço IP de destino. Quando o destino recebe este pacote ele envia um pacote ICMP de resposta do tipo *echo reply* de volta a origem. Quando a origem recebe esta mensagem de resposta, ela calcula o tempo que o pacote demorou para ir e voltar e mostra este tempo na tela. Este valor é conhecido como RTT (*Round Trip Time*).

São descritas as três possibilidades de resultado:

O pacote vai e volta com sucesso

Neste caso tudo ocorreu como deveria e portanto há conectividade entre os dois nós.

O pacote vai, mas não há resposta do outro lado

Neste caso o pacote conseguiu chegar ao destino, mas ocorreu algum problema no percurso do pacote de resposta (*echo reply*). Esta situação pode ser observada quando o comando ping não escreve nada na tela, pois está aguardando algum pacote de resposta. Em geral esse problema é devido a alguma configuração de rota ou IP incorretos.

O pacote não consegue chegar ao destino

O principal indício para este caso é a aparição da mensagem *Destination unreachable*: o pacote não consegue nem chegar à máquina de destino, pois ela não tem informações suficientes sobre como chegar a esse destino. Neste caso usualmente há algum problema de rotas ou configuração de IP em algum lugar da rede.

Captura de pacotes por meio do tcpdump

Para capturar os pacotes que trafegam em uma determinada interface, é utilizado comando tcpdump:

```
# tcpdump -i <nome da interface> -s 0 -w <arquivo de captura>
```

Desta forma, o comando irá armazenar os dados coletados no <arquivo de captura>. Caso queira que as informações sejam disponibilizadas na tela, basta omitir o parâmetro w:

```
# tcpdump -i <nome da interface> -s 0
```

Para encerrar a captura, basta utilizar a combinação de teclas Ctrl+C.

Análise de pacotes por meio do Wireshark

O Wireshark é um analisador de pacotes capaz de realizar captura em tempo real ou leitura de um arquivo gerado por meio do tcpdump. Para utilizar o Wireshark basta executá-lo no Desktop, como apresentado na Figura C.2.

1. Captura com gravação de arquivo.

Vá em File -> Open e selecione o arquivo que deseja abrir (neste caso serão arquivos de sufixo .pcap), conforme a Figura C.3.

O Wireshark deve mostrar os pacotes que foram capturados durante a execução do tcpdump como mostrado na Figura C.4.

2. Captura em tempo real com resultados em tela.

Para iniciar uma captura em tempo real por meio do Wireshark, vá em Capture>start, conforme apresentado na Figura C.5.

Caso queira parar uma captura basta ir em Capture>stop, desta forma, o Wireshark não irá mais capturar os pacotes das interfaces.



Figura C.2: atalho para o Wireshark no Desktop.

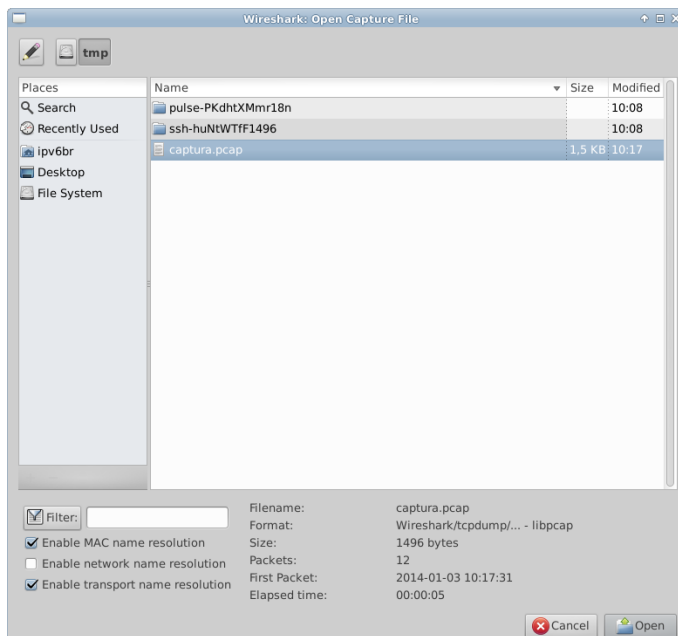


Figura C.3: arquivos de sufixo .pcap.

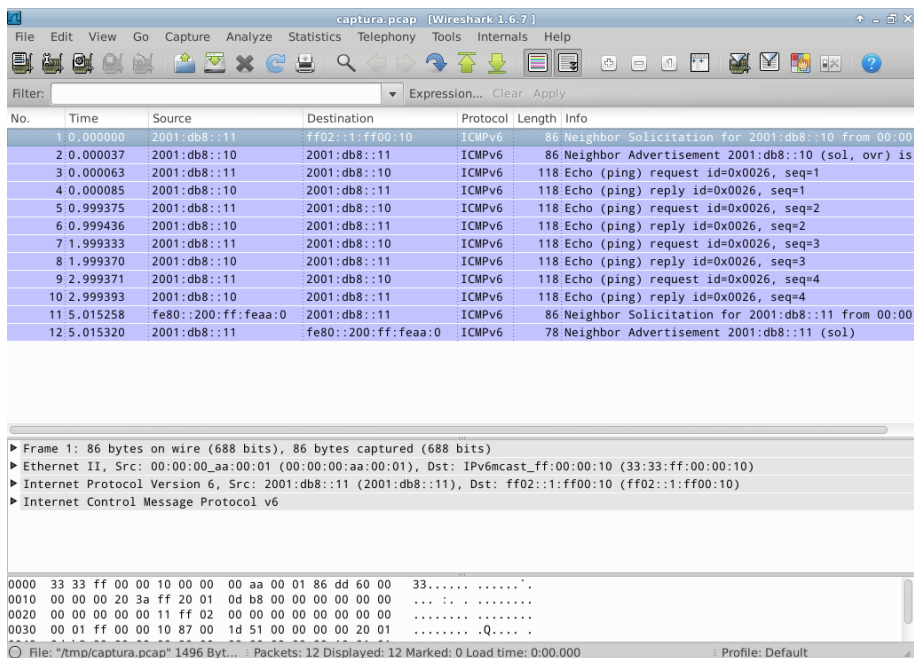


Figura C.4: conteúdo de um arquivo de sufixo .pcap exibido pelo Wireshark.

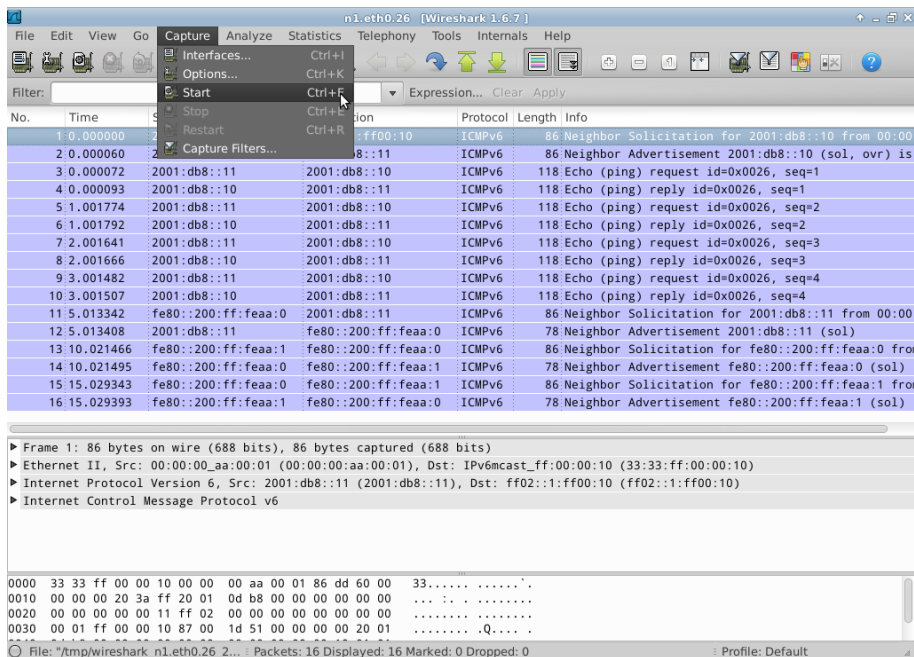


Figura C.5: captura ao vivo.

Editores de texto

Aqui será abordado dois editores de texto bastante conhecidos pela comunidade Linux. Ambos funcionam utilizando o terminal, permitindo que as alterações dos textos sejam feitas dentro do terminal do equipamento emulado no CORE.

nano

O nano é um editor de texto utilizando o terminal que possui algumas funções por meio dos atalhos `Ctrl+<comando>` ou das teclas F1 a F12.

Para utilizar o nano dentro do terminal basta utilizar:

```
# nano <nome do arquivo>
```

Alguns comandos importantes do nano:

`Ctrl+G` ou F1

Abre menu de ajuda

`Ctrl+O` ou F3

Salva o arquivo em edição

`Ctrl+W` ou F6

Busca por um trecho específico de texto

`Ctrl+X` ou F2

Sai do editor nano

vi

O vi (*visual editor*) é um editor de texto de padrão avançado, presente em qualquer distribuição Unix. O vi trabalha em dois modos distintos: *insert mode* e *command mode*.

Insert mode (Modo de inserção de texto normal)

Para ativá-lo basta pressionar o caractere `i`. Neste modo qualquer caractere digitado será inserido no texto no local em que o cursor se encontra. Note que no vi padrão, até mesmo teclas como as setas direcionais, *Ctrl*, *Backspace* e *Delete* vão escrever seu caractere correspondente no texto. Para realizar qualquer operação que não seja inserir caracteres no texto deve-se utilizar o modo comando.

Command mode (Modo de comandos do vi)

Para ativá-lo deve-se pressionar a tecla Esc de seu teclado. Para alterar a posição do cursor, utiliza-se as teclas h, j, k e l, que correspondem aos direcionais para a esquerda, abaixo, acima e à direita respectivamente.

Além disso, o vi possui uma série de comandos para ajudar na edição:

x Deleta o caractere indicado pelo cursor.

u Desfaz a última ação.

Para conhecer os demais comandos, visite <http://www.cs.rit.edu/~cslab/vi.html>

Saindo e salvando o arquivo no vi

Uma das maiores dificuldades dos iniciantes em vi é sair dele. Para sair do editor, deve-se digitar o comando :q seguido de Enter dentro do command mode. Caso queira forçar a saída do programa sem salvar as alterações feitas deve-se digitar o comando :q! seguido de Enter. Para salvar o arquivo em edição, deve-se utilizar o comando :w seguido de Enter.

Referências Bibliográficas

- Abley et al.(2007)** J. Abley, P. Savola, e G. Neville-Neil. Deprecation of Type 0 Routing Headers in IPv6. RFC 5095 (Proposed Standard), Dezembro 2007. <http://www.ietf.org/rfc/rfc5095.txt>.
- Arkko et al.(2005)** J. Arkko, J. Kempf, B. Zill, e P. Nikander. SEcure Neighbor Discovery (SEND). RFC 3971 (Proposed Standard), Março 2005. <http://www.ietf.org/rfc/rfc3971.txt>. Atualizada pelas RFCs 6494, 6495, 6980.
- Arkko et al.(2010)** J. Arkko, M. Cotton, e L. Vegoda. IPv4 Address Blocks Reserved for Documentation. RFC 5737 (Informational), Janeiro 2010. <http://www.ietf.org/rfc/rfc5737.txt>.
- Bagnulo et al.(2011)** M. Bagnulo, A. Sullivan, P. Matthews, e I. van Beijnum. DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers. RFC 6147 (Proposed Standard), Abril 2011. <http://www.ietf.org/rfc/rfc6147.txt>.
- Blanchet(2008)** M. Blanchet. Special-Use IPv6 Addresses. RFC 5156 (Informational), Abril 2008. <http://www.ietf.org/rfc/rfc5156.txt>. Obsoletada pela RFC 6890.
- Brijeski(2012)** Tony Brijeski. Remastersys, Setembro 2012. <http://www.remastersys.com/>. Último acesso em 21/10/2014.
- Canonical(2012)** Canonical. Xubuntu, Abril 2012. <http://www.xubuntu.org/>. Último acesso em 21/10/2014.
- Coltun et al.(2008)** R. Coltun, D. Ferguson, J. Moy, e A. Lindem. OSPF for IPv6. RFC 5340 (Proposed Standard), Julho 2008. <http://www.ietf.org/rfc/rfc5340.txt>. Atualizada pelas RFCs 6845, 6860, 7503.

- Davies e Mohacsi(2007)** E. Davies e J. Mohacsi. Recommendations for Filtering ICMPv6 Messages in Firewalls. RFC 4890 (Informational), Maio 2007. <http://www.ietf.org/rfc/rfc4890.txt>.
- Despres(2010)** R. Despres. IPv6 Rapid Deployment on IPv4 Infrastructures (6rd). RFC 5569 (Informational), Janeiro 2010. <http://www.ietf.org/rfc/rfc5569.txt>.
- Dommety(2000)** G. Dommety. Key and Sequence Number Extensions to GRE. RFC 2890 (Proposed Standard), Setembro 2000. <http://www.ietf.org/rfc/rfc2890.txt>.
- Durand et al.(2011)** A. Durand, R. Droms, J. Woodyatt, e Y. Lee. Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion. RFC 6333 (Proposed Standard), Agosto 2011. <http://www.ietf.org/rfc/rfc6333.txt>. Atualizada pela RFC 7335.
- Farinacci et al.(2000)** D. Farinacci, T. Li, S. Hanks, D. Meyer, e P. Traina. Generic Routing Encapsulation (GRE). RFC 2784 (Proposed Standard), Março 2000. <http://www.ietf.org/rfc/rfc2784.txt>. Atualizada pela RFC 2890.
- Hinden e Deering(2006)** R. Hinden e S. Deering. IP Version 6 Addressing Architecture. RFC 4291 (Draft Standard), Fevereiro 2006. <http://www.ietf.org/rfc/rfc4291.txt>. Atualizada pelas RFCs 5952, 6052, 7136, 7346, 7371.
- IANA(2014)** IANA. Service name and transport protocol port number registry, Outubro 2014. <http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml>. Último acesso em 21/10/2014.
- Huston et al.(2004)** G. Huston, A. Lord, e P. Smith. IPv6 Address Prefix Reserved for Documentation. RFC 3849 (Informational), Julho 2004. <http://www.ietf.org/rfc/rfc3849.txt>.
- Ishiguro(2006)** Kunihiro Ishiguro. IPv6 support, Julho 2006. <http://www.nongnu.org/quagga/docs/docs-info.html#SEC140>. Último acesso em 13/8/2012.
- Kent e Atkinson(1998)** S. Kent e R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard), Novembro 1998. <http://www.ietf.org/rfc/rfc2401.txt>. Obsoletada pela RFC 4301, atualizada pela RFC 3168.

- Kent e Seo(2005)** S. Kent e K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dezembro 2005. <http://www.ietf.org/rfc/rfc4301.txt>. Atualizada pela RFC 6040.
- McCann et al.(1996)** J. McCann, S. Deering, e J. Mogul. Path MTU Discovery for IP version 6. RFC 1981 (Draft Standard), Agosto 1996. <http://www.ietf.org/rfc/rfc1981.txt>.
- Naval Research Laboratory(2009)** Naval Research Laboratory. Common open research emulator (CORE), Julho 2009. <http://cs.itd.nrl.navy.mil/work/core/>. Último acesso em 21/10/2014.
- Naval Research Laboratory(2012)** Naval Research Laboratory. Quagga mobile routing 0.99.21mr2.2 for linux, Julho 2012. <http://downloads.pf.itd.nrl.navy.mil/ospf-manet/quagga-0.99.21mr2.2/>. Último acesso em 21/10/2014.
- Naval Research Laboratory(2013)** Naval Research Laboratory. CORE package 4.5 for linux, Março 2013. <http://downloads.pf.itd.nrl.navy.mil/core/packages/4.5/>. Último acesso em 21/10/2014.
- Nikander et al.(2007)** P. Nikander, J. Laganier, e F. Dupont. An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID). RFC 4843 (Experimental), Abril 2007. <http://www.ietf.org/rfc/rfc4843.txt>. Obsoletada pela RFC 7343.
- Nordmark e Gilligan(2005)** E. Nordmark e R. Gilligan. Basic Transition Mechanisms for IPv6 Hosts and Routers. RFC 4213 (Proposed Standard), Outubro 2005. <http://www.ietf.org/rfc/rfc4213.txt>.
- Oracle(2015)** Oracle. VirtualBox, Fevereiro 2015. <https://www.virtualbox.org/>. Último acesso em 02/02/2015.
- Popoviciu et al.(2008)** C. Popoviciu, A. Hamza, G. Van de Velde, e D. Dugatkin. IPv6 Benchmarking Methodology for Network Interconnect Devices. RFC 5180 (Informational), Maio 2008. <http://www.ietf.org/rfc/rfc5180.txt>.
- Rekhter et al.(1996)** Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, e E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), Fevereiro 1996. <http://www.ietf.org/rfc/rfc1918.txt>. Atualizada pela RFC 6761.

Rekhter *et al.*(2006) Y. Rekhter, T. Li, e S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), Janeiro 2006. <http://www.ietf.org/rfc/rfc4271.txt>. Atualizada pelas RFCs 6286, 6608, 6793.

Savola(2011) Pekka Savola. `radvd.conf(5)` - linux man page, Janeiro 2011. <http://linux.die.net/man/5/radvd.conf>. Último acesso em 21/10/2014.

Thomson *et al.*(2003) S. Thomson, C. Huitema, V. Ksinant, e M. Souissi. DNS Extensions to Support IP Version 6. RFC 3596 (Draft Standard), Outubro 2003. <http://www.ietf.org/rfc/rfc3596.txt>.

Índice remissivo

Símbolos

464XLAT, 310

4in6, 268, 270, 272

6in4, 255, 262, 276, 283

6rd, 276, 280, 281

A

acesso

Web, 147

Address Family Transition Router,
veja AFTR

address-family, 340

Advertise, 52, 60, 90

AFTR, 268–270

AH, 217, 220, 235, 243

allow-query, 119

allow-query-cache, 109, 114

allow-recursion, 110, 114

Apache, 130, 148, 160

ARP, 9, 186

ARPANet, 1

AS, 334

as-path, 357

ASN, 334, 339

autenticação, 218, 242

autenticar, 219

autenticidade, 217, 218

Authentication Header, *veja AH*

Autonomous Systems, *veja AS*

Autonomous Systems Number, *veja*
ASN

B

B4, 268

Basic Bridging Broad Band, *veja*
B4

BGP, 334, 338, 343, 345, 350, 358

BIND, 41, 105, 110, 295

BIND9, *veja BIND*

Border Gateway Protocol, *veja BGP*
browsers, *veja navegadores*

C

cache

DNS, 108, 120

proxy Web, 147, 160, 164, 167

Carrier Grade NAT, *veja CGN*

ccTLD, 106

certificado X.509, 218

CGN, 268, 269

CGNAT, 5

chave

AH, 218, 220

autenticação, 220, 242

criptografia, 220

ESP, 218, 220

ISAKMP SA, 218

pública, 199

pré-compartilhada, 218

CIDR, 4

CIFS, 168

CLAT, 311

clear bgp, 350, 358

- Common Internet File System, veja*
 CIFS
 confidencialidade, 217
 CORE, 7, 367
country code top level domains, veja
 ccTLD
 CPE, 82, 268, 276, 277, 311
 criptografar, 219
 criptografia, 217, 242
Customer Premises Equipment, veja
 CPE
Customer Side Translator, veja
 CLAT
- D**
 DAD, 27, 30, 178, 181
Default Routers, 52
Denial of Service, veja negação de
 serviço
 descoberta de roteadores, 15, 22
 descoberta de vizinhança, *veja*
 NDP
 descoberta de vizinhos, *veja* NDP
destination cache, 100
 detecção de endereços duplicados,
veja DAD
 DHCP, 4, 51, 281
 DHCPv4, 52
 DHCPv6, 51, 52, 81
disable-empty-zone, 110
 DNS, 52, 56, 60, 62, 105, 106, 281,
 308
 autoritativo, 108, 117
 recursivo, 62, 66, 106, 108
 DNS64, 294
 domínios de primeiro nível, 106
Domain Name System, veja DNS
 DoS, *veja* negação de serviço
 DS-Lite, 268
Dual Stack Lite, veja DS-Lite
 dupla tradução, 311
- Duplicate Address Detection, veja*
 DAD
 duplo NAT, 269
*Dynamic Host Configuration Pro-
 tocol, veja* DHCP
- E**
 eBGP, 342, 343
echo reply, 382
echo request, 382
 Emulador de Redes, 367
 encapsulamento, *veja* encapsular
 encapsular, 255, 256, 259, 262, 265,
 268, 271, 276, 280, 281
Encapsulated Security Payload, veja
 ESP
 endereço físico, *veja* endereço MAC
 endereço IP, 2, 3
 endereço MAC, 9, 10, 12, 13, 15, 27,
 60, 62, 197, 375
 endereço não especificado, 27
 enlace, 9, 15, 35, 99, 147, 182, 197,
 256, 323
 ESP, 217, 220, 235
 extensões multiprotocolo do BGP,
veja MP-BGP
- F**
firewall, 185
 IPv4, 186
 IPv6, 186
 Flag, 14
*Autonomous Address-
 Configuration*, 40, 49,
 76, 97
*Managed Address Configura-
 tion*, 53
Other Configuration, 53, 67
 fragmentação, 99, 103, 256
 FreeBSD, 295

G

gateway, 17, 318
Generic Routing Encapsulation,
 veja GRE
generic top level domains, veja
 gTLD
 GRE, 262
 gTLD, 106

H

hexadecimal, 107, 224, 232, 233,
 255, 262, 281, 290, 301, 304,
 308
hosts allow, 171
 HTML, 130
 HTPC, 152, 164
 HTTP, 130, 133, 137, 138, 144, 150,
 162, 199

I

iBGP, 338, 342
 IBM, 168
 ICMPv4, 186, 256, 272
 ICMPv6, 99, 177, 186, 196, 197, 199,
 212, 256
 ICP, 152, 164
 ifconfig, 30, 181, 380
 IGMP, 186
 IGP, 323
 IKE, 218
Information-Request, 77, 78
 integridade, 217
 Intel, 168
Interior Gateway Protocol, veja
 IGP
 Internet, 1, 81, 99, 106, 147, 217,
 256, 293, 310, 311, 334, 335,
 338
Internet Cache Protocol, veja ICP
 Internet das Coisas, 5
Internet Key Exchange, veja IKE

*Internet Security Association Key
 Management Security
 Association*, veja chave
 ISAKMP SA

Internet Service Provider, veja ISP
 ip6.arpa, 107
 ip6tables, 199, 213
 iproute2, 380
 IPsec, 217, 218
 iptables, 293, 299
 IPv4, 3, 81, 99, 117, 178, 186, 217
 IPv6, 3, 10, 30, 81, 99, 100, 138, 178
IPv6 Rapid Deployment, veja 6rd
 ISIS, 262
 ISP, 269, 277, 280–282, 295, 297,
 304, 305, 311, 315
 IVI, 310, 311

K

kernel, 294, 319

L

link-local, 15, 35, 52, 263, 381
 Linux, 6, 30, 148, 182, 259, 265, 280,
 281, 294, 295, 319, 323, 334,
 363, 386
listen-on, 119
listen-on-v6, 114, 122
local-preference, 358, 360, 361

M

máquina virtual, 363
 Mac OS, 6
man-in-the-middle, 245
 mecanismos de transição, veja téc-
 nica de transição
 Microsoft, 168
 Microsoft IIS, 148
 modo de transporte, 217, 219
 modo túnel, 217, 244
 MP-BGP, 335
 mtr, 129

MTU, 99, 100
multicast, 11, 100
 all-dhcp-agents, 52
 all-nodes, 15
 all-routers, 15, 211
 solicited-node, 10, 180
multiprotocol BGP, veja MP-BGP

N

NA, 10, 27, 31, 178, 181, 196, 197
name server, 305
name-servers, 55
 nano, 386
 NAT, 4, 268, 269, 293, 294, 299
 NAT44, veja NAT
 NAT46, 310, 317
 NAT64, 293, 294, 310–312, 314, 315
 NATv4, veja NAT
 navegadores, 130
 NDP, 9, 15, 27, 52, 53, 177, 186
 NDPMon, 182, 183
 negação de serviço, 177, 182, 245
Neighbor Advertisement, veja NA
neighbor cache, 10
Neighbor Discovery, veja NDP
Neighbor Discovery Protocol, veja NDP
Neighbor Solicitation, veja NS
 Nginx, 160
 nomes de domínios, 105, 106, 124, 165, 304
 NS, 10, 27, 31, 178, 181, 196

O

OSPF, 323
 OSPFv2, 324
 OSPFv3, 324
overhead, 262

P

página
 Web, 106, 130, 137, 138

packet too big, 99, 212, 213, 256
Path MTU Discovery, veja PM-TUD

payload, 255, 262
 pilha dupla, 147, 160, 311, 324
 ping, 116, 129, 381
 ping6, 102, 116, 129, 381
 PLAT, 311
 PMTUD, 99, 212
 políticas
 AS-IN, veja políticas de entrada
 AS-OUT, veja políticas de saída
 bloqueio de ICMP, 186
 de entrada, 335, 353, 356, 358, 361
 de roteamento, 335, 347
 de saída, 335, 347, 348, 350
 de segurança, 224, 225, 227, 229, 235
 FORWARD, 200
 INPUT, 200
 OUTPUT, 200
 SPF, 123

Prefix Delegation, 81
prefix-list, 349, 350, 357, 358
 protocolo 41, 255, 280, 281
 provedor
 de acesso à Internet, 81, 106, 268, 276, 277, 285
 de serviços, 334
 de trânsito, 334, 342, 347
Provider Side Translator, veja PLAT

Q

Quagga, 35, 71, 323, 334

R

RA, 15, 17, 35, 53, 67, 71

- radvd, 41, 45, 85
 - RARP, 9, 186
 - rdnssd, 43
 - recursion*, 119
 - redistribute connected*, 328
 - Registro
 - A, 106, 107, 293
 - AAAA, 106, 107, 293
 - CNAME, 106
 - MX, 106
 - NS, 106
 - PTR, 106, 107
 - quad-A, 107
 - SOA, 106
 - Registro de Recursos, 105
 - Relay 6rd*, 276
 - Reply*, 52
 - Request*, 52
 - resiliente, 334
 - resolver*, 106
 - Resource Records*, veja Registro de Recursos
 - reverso
 - proxy*, 160, 161, 163
 - resolução, 107
 - rota estática, 259, 265, 343, 349
 - rota padrão, 41, 54, 287, 313
 - roteador de borda, 293, 294, 296, 311
 - roteador padrão, 52, 54
 - roteamento, 311, 324, 335
 - tabela BGP, 338
 - tabela de, 331, 350
 - tomada de decisão de, 353
 - route-map*, 350, 357
 - Router Advertisement*, veja RA
 - router-id*, 328, 339
 - RR, veja Registro de Recursos
 - RS, 15
 - RTT, 382
- S**
- SA, 223, 224, 227
 - SAD, 223
 - Samba, 168, 171, 173–175
 - Secure Neighbor Discovery*, veja SEND
 - Security Association*, veja SA
 - Security Association Database*, veja SAD
 - Security Policy Database*, veja SPD
 - segurança, 217
 - SEND, 182
 - Server Message Block*, veja SMB
 - servidor, 168
 - autoritativo, 106, 108, 117, 119
 - DHCP, 51
 - DHCPv6, 52, 53, 67, 82
 - DNS, 41, 52, 67, 71, 73, 112
 - DNS64, 293, 304, 305, 308
 - Nginx, 141, 146
 - proxy*, 147, 152
 - raiz, 106
 - recursivo, 106, 117, 293, 304
 - Samba, 170
 - SIP, 52
 - Squid, 166
 - Web*, 130, 148, 160, 163
 - SIP, 52
 - Sistema Autônomo, veja AS
 - SLAAC, veja autoconfiguração *stateless*
 - SMB, 168
 - smurf*, 245
 - SNMP, 281
 - Solicit*, 52
 - SPD, 223, 227
 - Squid, 148, 151, 160, 163
 - SSH, 199
 - stateful*
 - 464XLAT, 311, 312

DHCP, 52, 81

firewall, 185

NAT64, 293

stateless

464XLAT, 310–312

autoconfiguração, 35, 45, 52,
69, 81, 277

DHCP, 52, 71, 73

firewall, 186

T

técnica de transição, 147, 160, 268,
270, 293, 310

túnel

6rd, *veja* 6rd

IPv4, *veja* 6in4

automático, 268

estático, 256

GRE, *veja* GRE

IPv4 sobre IPv6, *veja* 4in6

manual, 255, 256

Target Link-Layer Address, 14

TAYGA, 293, 294, 310, 312, 313

tcpdump, 383

THC-IPv6, 180

Totd, 295

traceroute, 129

traceroute6, 129, 199, 333, 347, 354

tradução, 293, 298, 310, 317, 320

de pacotes, 296, 310

nomes de domínio, 105

protocolos IP, 293, 294, 311,
312, 321

U

Unix, 35, 168, 367, 386

V

vi, 386

Virtual Machine, *veja* máquina virtual

VirtualBox, 363

VirtualServer, 130, 146

VM, *veja* máquina virtual

vttysh, 328, 339

W

Windows, 6, 295

Wireshark, 383

“Como o próprio nome indica, este livro tem um caráter prático e contém roteiros para experimentos que podem auxiliá-lo no seu aprendizado. Ele pode ser usado tanto por quem está começando a aprender sobre redes agora, como por profissionais experientes. Não é um livro apenas para ler, você deve realizar os experimentos. (...)”

A equipe do IPv6.br – o projeto de disseminação do IPv6 do NIC.br – foi quem preparou e aperfeiçoou estes experimentos. Tais experimentos tem sido empregados, com muito sucesso, nos cursos de formação do NIC.br. Centenas de alunos e de profissionais já seguiram estes mesmos roteiros. Eles comprovadamente ajudam a entender a forma como o IPv6 funciona, como se diferencia do protocolo IPv4, e como realizar configurações na prática em uma série de situações. Os experimentos proporcionam uma excelente base prática, que o ajudará muito no seu dia a dia.”

Tereza Cristina Melo De Brito Carvalho

Profa. Associada da Escola Politécnica da USP

Coordenadora técnica de projetos

do LARC-PCS-EPUSP

(no prefácio)

Aprenda na prática

- Funcionalidades básicas do IPv6
- Autoconfiguração de endereços e DHCPv6
- Configuração de servidores DNS, HTTP, proxy e de arquivos
- Configuração de firewall e IPSec
- Técnicas de transição: 6in4, GRE, DS-Lite, NAT64
- Roteamento OSPF e BGP

Fique conectado:



twitter.com/novateceditora



facebook.com/novatec



www.novatec.com.br