ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de Sistemas Computacionais

6IX LINUX: DIVULGANDO E INTENSIFICANDO A UTILIZAÇÃO DO NOVO PROTOCOLO DA INTERNET (IPv6)

Trabalho de Conclusão de Curso

Engenharia da Computação

Pedro França Lima Neto
Orientador: Prof. Doutor Adriano Lorena Inácio de Oliveira

Recife, 18 de novembro de 2005

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de Sistemas Computacionais

6IX LINUX: DIVULGANDO E INTENSIFICANDO A UTILIZAÇÃO DO NOVO PROTOCOLO DA INTERNET (IPv6)

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Pedro França Lima Neto
Orientador: Prof. Doutor Adriano Lorena Inácio de Oliveira

Recife, 18 de novembro de 2005

UNIVERSIDADE
DE PERNAMBUCO

Pedro França Lima Neto

6IX LINUX: DIVULGANDO E INTENSIFICANDO A UTILIZAÇÃO DO NOVO PROTOCOLO DA INTERNET (IPv6)

Resumo

O protocolo IP utilizado hoje, o chamado IPv4, não foi projetado para uma quantidade tão grande de máquinas interconectadas em uma mesma rede comum, como é o caso da Internet. O IPv4 possui um número limitado de endereços, e uma capacidade de multiplicação restrita. Para tentar evitar a estagnação da Internet, ainda em franca expansão, teriam de ser criadas soluções para evitar a todo custo a provável falta dos endereços IP. Mas as medidas adotadas só fizeram retardar o prazo previsto para exaustão dos endereços, o que acabou levando à criação de uma solução definitiva para o problema: o IPv6. A capacidade de endereçamento do IPv6 é um número virtualmente impossível de ser totalmente alocado, mesmo a longo prazo, e ainda incorpora soluções para várias questões surgidas durante a trajetória da Internet, e que tiveram que ser resolvidas com o IPv4 de forma emergencial, e muitas vezes ineficiente. É objetivo deste trabalho introduzir as principais características, mudanças e implicações acrescentadas pelo surgimento desse novo protocolo da internet, o IPv6, e apresentar um sistema operacional, o 6ix Linux, que conta com a ajuda de várias ferramentas e utilitários específicos para configuração do acesso e comunicação IPv6, através da técnica de transição conhecida como tunelamento. Todo o processo de criação e desenvolvimento da distribuição 6ix Linux é descrito em detalhes, bem como a avaliação dos resultados de experiências realizadas com o intuito de comparar e analisar os dois métodos de tunelamento presentes neste sistema operacional. A intenção principal na criação do 6ix Linux é divulgar e massificar a utilização do novo protocolo, familiarizando os usuários e desmistificando conceitos atualmente compreendidos apenas por especialistas, ajudando a acelerar a migração para o protocolo IPv6.

Abstract

The currently used IP protocol, known as IPv4, was not designed for a so great number of machines interconnected in a common network, such as the Internet. IPv4 has limited number of addresses and restricted capacity of multiplication. Trying to prevent the stagnation of the Internet, still in expansion, it would be necessary to create solutions preventing, by some way, the lack of IP addresses. The adopted actions had only delaied the foreseen stated period for exhaustion of the addresses. This has motivated the proposal of a definitive solution for the problem: the IPv6. IPv6's capacity of addressing is a number virtually impossible to be totally assigned, even in the long run, and it also incorporates solutions for some questions that appeared during the trajectory of the Internet. Such questions had to be urgently treated, many times inefficiently, with IPv4. This work aims to introduce the main characteristics, changes, and implications added by the arrive of this new Internet Protocol and to present an operational system, the 6ix Linux, that provides some specific auxiliary softwares and tools for configuration of IPv6 access and communication, through the transition technique known as tunneling. All the creation and development process of the distribution 6ix Linux is described in details, as well as the evaluation of the results of experiences carried through with intention to compare and to analyze the two methods of tunneling provided in this operational system. The main intention of 6ix Linux is to help spread the use of the new protocol, making it more familiar to the users and demystifying concepts currently understood only by specialists, helping to speed up the migration to IPv6.

Sumário

Índice de Figuras

Índice de Quadros

Agradecimentos

Agradeço a meus pais, José Alberto e Maria Leônia, por todo o amor, carinho e apoio que sempre me deram, me guiando e me incentivando a galgar dificuldades e fazendo das tripas coração para bancar os meus estudos.

A meus padrinhos, Antônio Tiago e Ivone Maria, por terem sido meus segundos pais durante a minha trajetória na faculdade, cedendo-me moradia e amor, deixando-me ocupar um espaço em seus corações e ensinando-me a agir com responsabilidade.

A minha tia Iana, por me permitir fazer de sua casa uma moradia aconchegante, nos momentos de angústia e de dúvida, conduzindo-me com suas conversas francas e palavras sábias.

Aos meus primos, por serem meus melhores amigos, e à minha família como um todo, por serem tão unidos e participarem de minha vida de forma tão presente.

Agradeço também à família de minha esposa, em especial minha sogra Maria do Socorro e minha cunhada Catalina, por serem tão generosas e contribuírem da melhor forma possível para comigo, minha esposa e meu filho.

Agradeço ainda a meus mestres, por todo o conhecimento compartilhado e pelos ensinamentos repassados de maneira tão natural e eficaz, tornando-me um profissional capacitado. Em especial, agradeço a Carlos Alexandre, por ter sido sempre tão exigente comigo e com meus colegas, nos preparando para encarar de frente quaisquer desafios, e por ter sido também um bom amigo. Também agradeço com destaque a meu professor orientador, Adriano Lorena, por ter acreditado em mim e me auxiliado de forma tão preocupada e prestativa.

Aos colegas de classe, sem excessões, pelos momentos de descontração e bagunça, pela troca de experiências, e também pelas “rodadas” de estudo realizadas tantas vezes em minha residência nas vésperas de provas.

Aos companheiros de trabalho, José Luciano e Sérgio, da FISEPE, e Laureano, Hítalo e Ana, do TRF, por terem sido tão prestativos em minha aprendizagem e evolução profissional durante os períodos de estágio.

Agradeço a todos – familiares, amigos, conhecidos, colegas de faculdade e de trabalho, etc. – que de alguma forma participaram de minha vida e contribuiram em minha formação pessoal e profissional, e cujos nomes não me vêm à cabeça neste momento. Afinal, são tantos.

Por fim, agradeço àqueles que hoje me dão mais força para continuar crescendo e me aperfeiçoando: minha esposa, Camilla Izabella, e meu primogênito, Braz Neto. Milla, obrigado pela paciência e pelo apoio. Sei que não deve ter sido fácil aguentar um companheiro tão estressado e ausente como tive que ser durante boa parte deste projeto. Braz, obrigado por existir e me proporcionar exercer a paternidade com momentos tão felizes.

Com este trabalho, encerra-se mais um ciclo em meu caminho, abrindo as portas para novas etapas a serem percorridas e vencidas, etapas estas que espero ultrapassar contando sempre com o apoio de todos que me cercam.

Do fundo do meu coração, OBRIGADO!

Capítulo 1

Introdução

Há cerca de quatro anos, a *Internet Assigned Numbers Authority* (IANA) [54], instituição responsável pelo sistema geral de registro de domínios no mundo, vem alertando a população e os pesquisadores sobre a iminente falta de números IP válidos, chegando mesmo a afirmar que a total escassez deverá acontecer dentro de alguns poucos anos. Para atestar suas afirmações, o IANA acabou por realizar uma série de estudos, os quais indicaram que até o presente foram consumidos cerca de 66% dos endereços disponíveis no mundo [3][22].

A falta de endereços IP atravancaria o crescimento da Internet, estagnando o número de máquinas conectadas, e impediria a agregação de novos usuários e novos serviços à grande rede mundial, frustrando uma grande parcela da população.

1.1 Considerações iniciais e motivação

É importante que se saiba o porquê dessa situação estar acontecendo. O fator principal é que o protocolo IP utilizado hoje como padrão, o chamado IPv4, nasceu no início da década de 80, tendo como finalidade principal se tornar um padrão de comunicação entre as diferentes redes existentes até aquela época. Mas, impulsionadas pela adoção dessa padronização, as redes de computadores se expandiram inesperadamente, culminando na popularização da Internet e na aparição da *World Wide Web*, esta criada por volta de 1995. O problema é que o IPv4 não foi projetado para uma quantidade tão grande de máquinas interconectadas em uma rede comum. Este protocolo possui um número limitado de endereços, hoje distribuídos entre mais de 600 milhões de usuários mundiais, e uma capacidade de multiplicação restrita, oferecendo suporte a aproximadamente um máximo teórico de 4,3 bilhões de endereços.

A propagação da Internet – e de serviços como a *web* e o correio eletrônico, portanto, surpreendeu os projetistas do IPv4, que, em conjunto com as várias instituições e órgãos de controle e de regulação das redes de computadores de todo o mundo, como o IANA e a *Internet Engineering Task Force* (IETF) [56], se viram obrigados a criar soluções para evitar a todo custo a provável falta dos endereços IP.

E novas idéias surgiram, e vêm sendo adotadas hoje. São soluções de toda sorte de complexidade e de criatividade, indo desde o simples racionamento de endereços por parte dos órgãos distribuidores de IP – que antes de liberar novos endereços, efetuam uma bateria de perguntas à instituição que os deseja obter, como “que iniciativas tomou para reduzir o volume de

endereçamentos que já possui?" – até soluções mais arrojadas, como a utilização de máscaras de sub-rede e o *Network Address Translation* (NAT).

No entanto, é sabido que essas soluções não são totalmente eficazes e tampouco suficientes. A grande rede mundial ainda continua se expandindo, e a adoção dessas soluções de emergência não tem acompanhado o seu ritmo. Em verdade, elas até atrapalham o desenvolvimento de novas tecnologias. A adoção do NAT, por exemplo, elimina comunicações fim-a-fim [7].

Então, já que as soluções existentes não serão capazes de evitar a exaustão dos endereços IPv4, foi preciso desenvolver uma nova alternativa. A resposta encontrada pelos pesquisadores, foi criar um novo protocolo de internet: o IP *Next Generation* (IPng).

Estudos nessa área já vinham sendo realizados muito antes dos anúncios publicados pelo IANA. As ameaças do órgão só contribuíram para a aparição de um consenso de que um novo protocolo deveria ser rapidamente adotado. No início da década de 90, antes mesmo da aparição da *World Wide Web*, uma grande quantidade de novos protocolos já vinha sendo proposta por vários pesquisadores de todo o mundo [24]. Entre 1991 e 1993 foram submetidas aos órgãos reguladores várias RFCs (*Requests for Comments*), até que, em 25 de julho de 1994 [61], foi discutido e aprovado pela IETF o novo modelo de endereçamento que seria adotado como padrão do novo protocolo: o IPv6.

Com um cabeçalho simplificado e com uma capacidade teórica de endereçamento de cerca de $3{,}4 \times 10^{38}$ números IP – bem mais do que os $4{,}3 \times 10^{9}$ possíveis endereços do IPv4, muito tem se exaltado o IPv6. Na realidade, o IPv6 é muito mais que aumento de capacidade de endereçamento. São muitas as vantagens prometidas pelos entusiastas dessa tecnologia, como melhores tempos de resposta, maior segurança, maior mobilidade, entre outras melhorias [8][18][39][61]. Mas como só agora têm surgido as primeiras redes IPv6 comerciais, os benefícios reais de tal protocolo são ainda desconhecidos, à exceção do aumento da quantidade de possíveis endereços.

1.2 Objetivo

Buscando analisar a viabilidade da implantação do IPv6, diversas pesquisas têm sido realizadas. Analisar as suas características e reais benefícios é imprescindível. Afinal, dentro de poucos anos a humanidade entrará na era da convergência digital, onde haverá uma única rede integrando a TV digital interativa, equipamentos móveis, computadores e outros dispositivos. E essa nova era deverá ser comandada justamente pelo protocolo IPv6, justificando a necessidade da investigação de suas particularidades e aspectos agregados.

O que se pode afirmar, portanto, é que a adoção de um novo protocolo de rede, capaz de acomodar o crescente número de usuários da Internet, é um fato consumado. O que não se pode afirmar com exatidão é em quanto tempo se dará essa migração, que deverá ser efetuada aos poucos.

Aos poucos, pois os custos envolvidos com a transição entre os protocolos são muitos, e elevados. A Internet correria o risco de ser desativada por um longo intervalo de tempo. Além disso, equipamentos de *hardware* teriam que ser completamente – ou quase completamente – substituídos, exigindo grandes investimentos por parte das instituições e causando desperdício desses equipamentos.

Dessa forma, fica clara a necessidade da adoção de propostas de transição gradativas, a serem realizadas a longo prazo. Existem hoje algumas abordagens de migração lenta. As duas mais conhecidas estão descritas, originalmente, na RFC 2893: a pilha dupla e o tunelamento [18][57]. Existe ainda uma terceira abordagem, conhecida como NAT-PT [18].

É intenção deste trabalho analisar parte das funcionalidades e peculiaridades desse novo protocolo – o IPv6, tais como a estrutura de endereçamento e outras mudanças, tendo como foco principal a confecção de uma distribuição Linux *Live-CD Debian-based*, o 6ix Linux, capaz de permitir, de forma simples, que usuários dos mais variados níveis de conhecimento possam obter acesso ao *backbone* IPv6 já existente, através de túneis. Além do acesso IPv6 em si, a distribuição conta também com algumas das várias ferramentas que já possuem suporte ao novo protocolo da Internet.

1.3 Organização deste trabalho

No Capítulo 2, são abordadas definições relativas à pilha de protocolos da arquitetura de rede TCP/IP. Posteriormente, temos os conceitos acerca do Protocolo da Internet propriamente dito, em sua quarta versão, o IPv4, abordando com maiores detalhes as idéias necessárias à compreensão de seu funcionamento de um modo geral, e com foco nas características mais relevantes para o entendimento das mudanças a serem introduzidas pelo IPv6.

A seguir, no Capítulo 3, será apresentado o novo Protocolo da Internet, o IPv6, mostrando as suas principais características, mudanças e implicações em relação à versão atualmente utilizada, como o formato de endereçamento, alguns aspectos de roteamento e a forma como deve interagir com outros protocolos.

O Capítulo 4 é de suma importância, pois nele serão discutidas as questões relacionadas à migração entre as versões do protocolo IP, expondo os principais métodos adotados durante a fase de mudança do protocolo, com destaque para o tunelamento.

Enquanto isso, o Capítulo 5 faz um apanhado geral sobre alguns pontos vinculados à segurança da informação transmitida através do protocolo IPv6, buscando eliminar dúvidas comuns existentes acerca do nível de proteção do protocolo e também as com referência à substituição e à convivência de protocolos surgidos para impor maior segurança à comunicação IPv4.

No Capítulo 6, veremos em detalhes o desenvolvimento da distribuição 6ix Linux, abordando cada um dos passos executados para chegar até a sua versão final, bem como também mostrando as dificuldades encontradas durante o desenrolar de sua confecção.

Ainda no mesmo capítulo, serão demonstrados os resultados de experiências realizadas com o intuito de entender o funcionamento na prática do novo protocolo, e também objetivando a comparação de desempenho entre as formas de acesso à rede IPv6 disponibilizadas na distribuição 6ix Linux.

Por fim, temos o Capítulo 7, onde faremos as conclusões a respeito deste trabalho e do projeto desenvolvido, citando também as contribuições, além de indicar melhorias e trabalhos futuros para aprimorar a distribuição, tornando-a mais completa, mais robusta e mais confiável.

Capítulo 2

A Internet e o protocolo IPv4

A Internet trata-se de uma grande rede global e de acesso público, formada por um conjunto de redes privadas (sistemas autônomos) interconectadas, de maneira a facilitar compartilhamento de informações e possibilitando o oferecimento de serviços com os mais diversos fins. Para se obter acesso à Internet, utiliza-se como padrão a pilha de protocolos TCP/IP, um modelo de arquitetura dividido em camadas, do qual o IP (*Internet Protocol*) faz parte.

Antes da aparição da Internet, havia apenas pequenas redes locais privadas – as LANs (*Local Area Networks*), e redes privadas mais abrangentes – as WANs (*Wide Area Networks*), cada uma com suas arquiteturas e *hardwares* próprios, e protocolos de comunicação proprietários. Tais redes não possuíam a capacidade de se comunicarem entre si. Foi aí que o TCP/IP surgiu, com um novo conjunto de protocolos, buscando justamente suprir a falta de compatibilidade que havia entre as tantas redes heterogêneas.

A primeira dessas redes privadas, a ARPANET, surgiu no final da década de 60, quando alguns grupos de pesquisadores publicaram seus primeiros esboços sobre uma nova idéia para a forma de transmitir dados através dos mais variados canais de comunicação, a comutação de dados por pacotes. Até aquele momento, o que se utilizava era a comutação de circuitos, para a transmissão de voz através das redes telefônicas.

Mas a ARPANET era um projeto do Departamento de Defesa Americano (DoD), a sua divulgação era muito restrita, e o acesso de novos computadores à rede, rigorosamente controlado. Era natural, então que os excluídos criassem suas próprias arquiteturas de rede. E foi o que aconteceu: tanto nos Estados Unidos quanto na Europa, começaram a surgir inúmeras redes proprietárias, como a XNS, da Xerox; a SNA, da IBM; a EUNET, na Europa; dentre muitas outras.

Enquanto isso, os integrantes da NSF (*National Science Foundation*), uma agência federal criada pelo congresso norte americano para promover o progresso da ciência, resolveram fundar a sua própria rede, a CSNET, criada com o intuito de conectar os departamentos científicos das universidades americanas que não faziam parte da ARPANET. A CSNET foi a primeira rede baseada na pilha de protocolos TCP/IP, e impulsionada pelos seus serviços evoluiu de maneira muito mais rápida do que imaginado, obrigando a NSF a desenvolver uma rede de maior capacidade, a NSFNET, que em pouco tempo se tornou uma espécie de *backbone* de interligação entre redes regionais espalhadas por todo o território americano. Mais ainda, a NSFNET começou a se expandir por outros países, interligando as redes americanas a instituições canadenses e até européias.

Quando os inventores da ARPANET viram o sucesso da CSNET, resolveram também migrar a sua infra-estrutura para o protocolo TCP/IP. Outras redes privadas também começaram a implementar e implantar o suporte ao TCP/IP.

A ARPANET, já de posse dos novos protocolos, foi então conectada à NSFNET, mas acabou sendo extinta em 1990, enquanto a NSFNET continuava seu crescimento. A NSF se viu então pressionada, por várias organizações, a permitir a exploração comercial da sua rede, o que ocasionou um crescimento muito maior e mais intenso do que qualquer outro anterior.

A NSFNET se tornou grande demais para ser mantida pela NSF, que começou, então, a patrocinar *workshops*, estudos e discussões para encontrar uma forma plausível de migrar sua infra-estrutura para a indústria privada. Foi quando sugiram os conceitos de provedores de acesso, como NSPs (*National Service Providers*), NAPs (*Network Access Points*) e ISPs (*Internet Service Providers*) [24].

Todo o ano de 1994 foi dedicado à migração das redes regionais da NSFNET para redes comerciais e, em abril de 1995, a NSFNET foi oficialmente dissolvida, dando origem àquela que hoje é conhecida como a Internet, à qual milhões de pessoas e organizações espalhadas por todo o mundo têm acesso [26].

A história completa da Internet pode ser conferida de forma mais aprofundada em livros como o de Comer [7] ou de Kurose e Ross [24].

2.1 O TCP/IP

Seguindo o pensamento de Comer [7], o TCP/IP foi desenvolvido de maneira com que o usuário pense que a Internet é uma única rede virtual que interconecta todos os *hosts* – ou hospedeiros ou sistemas finais, e através da qual a comunicação é possível, enquanto sua arquitetura inferior é mantida oculta e é irrelevante para esses usuários finais.

A aparição da Internet praticamente só aconteceu devido à adoção da pilha de protocolos TCP/IP, pois graças a esse padrão tornou-se possível a interligação de redes (*internetworking* ou *internetting*), heterogêneas ou não, e com pequenas, médias ou grandes distâncias geográficas entre si – LANs e WANs. Seu nome se deve aos dois protocolos mais utilizados na transmissão dos dados, o TCP (Transmission Control Protocol) e o IP.

Originalmente criado pela ARPA, o TCP/IP logo passou a ser administrado e desenvolvido pelo IETF, um subgrupo de pesquisa integrante da *Internet Architecture Board* (IAB). A intenção era fazer com que o novo padrão se tornasse publicamente acessível, incentivando assim a sua adoção.

Hoje em dia, a IAB passou a se chamar *Internet Society* (ISOC) [62], e através do IETF continua, ainda, regulamentando e dando rumo não só às pesquisas desenvolvidas sobre o TCP/IP, como também sobre praticamente todos os padrões envolvidos com a Internet. As versões mais atuais de cada padrão são disponibilizadas publicamente, sem custo algum, através de relatórios técnicos chamados RFCs (*Request for Comments*) [57].

Além de ser público, o padrão TCP/IP tem como uma grande vantagem a sua organização em camadas [39]. Essa estrutura permite que a modificação de componentes de uma camada seja feita sem que se alterem componentes de outras camadas, tornando o padrão mais confiável, flexível e de fácil implementação. A idéia de camadas serve apenas para estabelecer relações hierárquicas entre os protocolos. Nesses moldes, cada camada inferior disponibiliza um serviço para a camada superior em um mesmo computador, e cada camada superior utiliza tais serviços para encaminhar os dados de uma origem até o seu destino.

O TCP/IP é composto de quatro camadas, constituídas por seus respectivos protocolos [7][24]. Na Figura 1, são mostradas as camadas e alguns de seus componentes. O meio de transmissão não é uma camada, é o canal por onde os dados trafegam fisicamente como, por exemplo, uma fibra ótica.

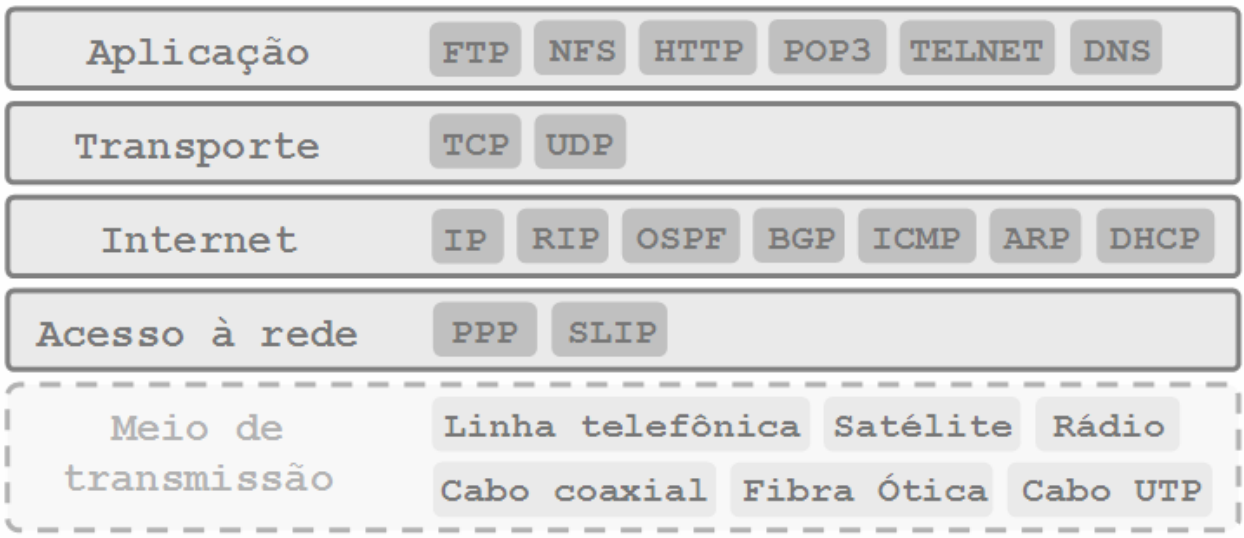

Figura 1. A pilha de protocolos TCP/IP.

Cada camada do TCP/IP acrescenta um cabeçalho aos dados da camada superior, contendo informações que são utilizadas pela camada equivalente no sistema final de destino. A esse processo, dá-se o nome de encapsulamento, e cada conjunto de dados mais o seu cabeçalho recebe a denominação genérica de PDU (*Protocol Data Unit*) da camada, podendo ser tratado de forma mais específica de acordo com a camada. A PDU da camada de Internet é mais conhecida como pacote, ou datagrama, enquanto a PDU da camada de acesso à rede é geralmente citada como *frame* (quadro).

2.2 O protocolo IPv4

Quando o padrão TCP/IP passou a ser adotado, a pilha de protocolos já havia sofrido uma série de alterações, como a melhor divisão das funções de cada camada, a criação de novos protocolos e a redefinição dos protocolos já existentes. Naquela época, os protocolos IP e UDP* (*User Datagram Protocol*) já haviam sido desmembrados do TCP, e a versão do protocolo IP já era a mesma da qual a Internet se utiliza atualmente, a versão 4 (IPv4).

O IPv4 é o integrante fundamental da camada de rede do TCP/IP, e dele dependem todos os protocolos de todas as camadas superiores, e mais alguns da própria camada de rede. Por esse

* O protocolo UDP, juntamente com o o TCP, constitui a camada de transporte. Esta camada não será abordada neste trabalho, no entanto é preciso frisar a sua importância. É nesta camada que se definem as portas utilizadas em uma comunicação. A porta é um número utilizado para identificar unicamente um serviço que se comunica pela rede, permitindo assim que várias aplicações sejam mantidas em execução de forma paralela no computador, compartilhando uma mesma interface de rede sem confundir os pacotes destinados a cada uma [7][25].

motivo, ele pode ser considerado o componente mais importante do padrão TCP/IP e, conseqüentemente, da Internet. Um verdadeiro pilar que sustenta toda a estrutura restante.

Existem três tipos de serviço principais na camada de rede: a montagem e endereçamento de pacotes; o roteamento de pacotes; e a troca de mensagens de controle e de erro. O roteamento, conhecido como um 'serviço de melhor esforço' por não prover garantias em relação à entrega dos dados [24], é realizado por um roteador, com a ajuda de protocolos de roteamento como o RIP (*Routing Information Protocol*), o BGP (*Border Gateway Protocol*) ou o OSPF (*Open Shortest Path First*). Já as mensagens de controle e de erro, são trocadas através do protocolo ICMP (*Internet Control Message Protocol*).

Para que sejam encaminhados corretamente, os pacotes de dados (datagramas) precisam de endereços que os identifiquem. O protocolo IPv4 é o responsável pela montagem desses datagramas IP e pelo endereçamento lógico de uma interface de rede, ligada a um *host* ou a outro equipamento de rede qualquer, permitindo aos dispositivos determinar o destino dos dados à medida que eles se movem pela rede. Também através do endereço IP é possível definir as máquinas e equipamentos que fazem parte de uma mesma rede corporativa, possibilitando a identificação dos mesmos, de forma hierárquica, até mesmo em outros sistemas autônomos.

2.2.1 O cabeçalho de pacotes IPv4

Cada pacote IP, também chamado de datagrama IP, é constituído de um conjunto de *bits*, e estes são agrupados em campos. O campo onde são colocados os dados da mensagem é chamado de campo de Dados. À exceção deste campo, todos os outros constituem o que se chama o cabeçalho do datagrama IP, e possuem finalidades específicas de controle e de roteamento.

A Figura 2 ilustra cada campo do datagrama IPv4, e a seguir vem uma breve descrição das funções exercidas por cada um desses campos, retiradas e resumidas a partir de Kurose [24].

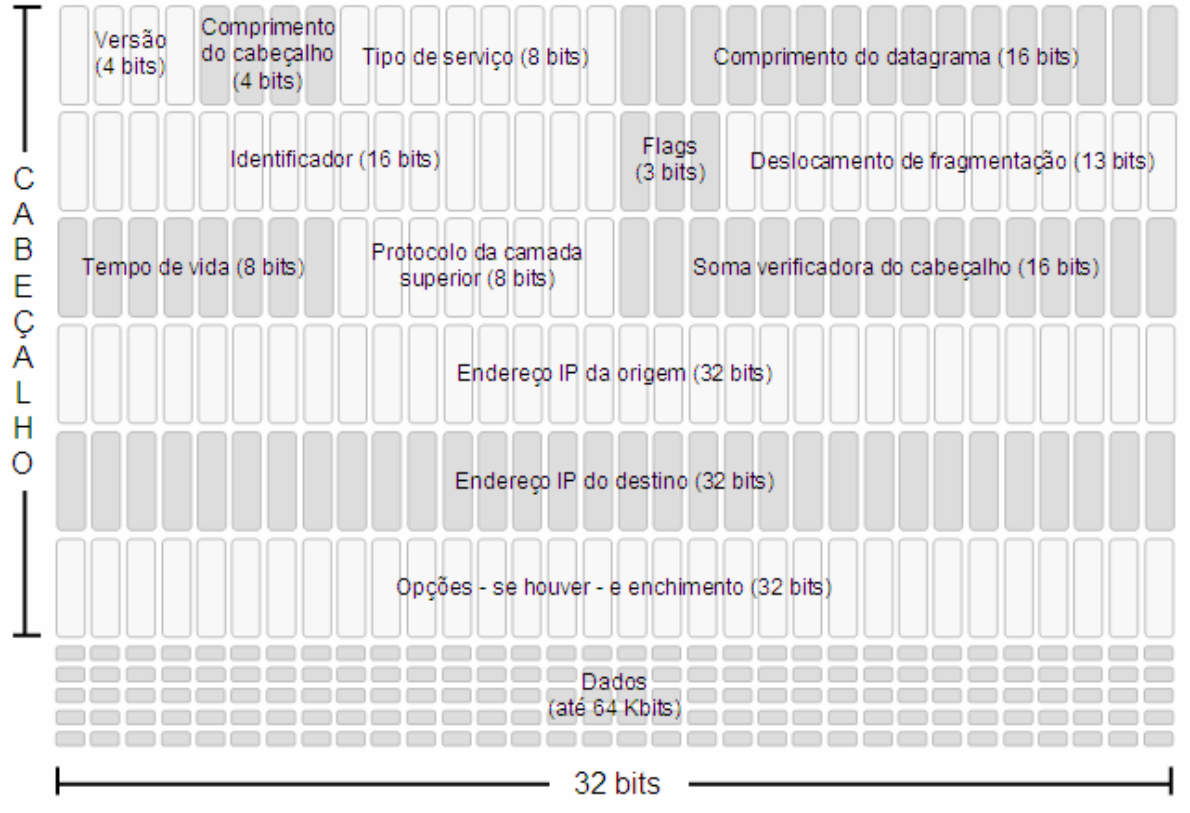

Figura 2. Os campos do datagrama IPv4.

Versão: indica a versão do protocolo IP utilizada pelo datagrama.

Comprimento do cabeçalho: indica o tamanho do cabeçalho do datagrama IP, que pode variar, mas que geralmente possui 20 *bytes*.

Tipo de serviço: distingue entre os diferentes tipos de serviços solicitados na camada superior, a fim de manipular os pacotes adequadamente.

Comprimento do datagrama: especifica o tamanho total do pacote IP, incluindo dados e cabeçalho.

Identificador, Flags, Deslocamento de fragmentação: Esses três campos estão relacionados à fragmentação de pacotes.

Tempo de vida: mantém um contador que diminui gradualmente, por incrementos, até zero, momento em que o datagrama é descartado, evitando que os pacotes permaneçam infinitamente em *loop*. Também conhecido como TTL (*Time to Live*).

Protocolo da camada superior: indica que protocolo de camada superior receberá os pacotes de entrada depois que o processamento do IP tiver sido concluído.

Soma verificadora do cabeçalho: auxilia na detecção de erros de *bits* em um datagrama IP recebido.

Endereços IP da origem e do destino: carregam, respectivamente, os endereços IP do remetente e do destinatário.

Opções e enchimento: raramente utilizado. Permite a extensão do cabeçalho IP, para acomodar suporte à segurança, por exemplo. Para assegurar que o cabeçalho seja sempre múltiplo de 32 *bits*, zeros são adicionados ao final do campo de opções, quando necessário.

Dados: contém os dados das camadas superiores e da mensagem propriamente dita.

2.2.2 O formato do endereçamento IPv4

Em sua concepção original, cada endereço IP de 32 *bits* era composto por duas partes conceituais principais, o número (prefixo) de rede e o número (sufixo) de *host*: o número de rede identificava a rede à qual pertencia o dispositivo, e o número de *host* identificava unicamente esse dispositivo dentro dessa rede, que poderia ser uma LAN, uma WAN ou mesmo a Internet [7][42].

Além disso, cada endereço IP pertencia a uma classe, *A*, *B* ou *C*, hoje chamadas classes cheias (*classfull*). Existiam, ainda, duas outras classes de IP, as classes *D* e *E*, que nunca estiveram comercialmente disponíveis. A classe *D* estava reservada para a utilização da tecnologia Multicast, e a classe *E* para pesquisas e uso futuro. A distinção entre as classes era feita através da configuração dos *bits* mais significativos do endereço. Devido à inclusão dos novos conceitos, propostos para tentar adiar a falta de IPs, os endereços exclusivamente baseados em classes deixaram de ser utilizados, passando a ocupar seu lugar os endereços sem classe (*classless*), juntamente com as máscaras de rede [7].

Para que a interpretação e configuração dos endereços IP se tornassem mais fáceis para os humanos, criou-se uma notação decimal separada por pontos, onde cada conjunto de 8 *bits* (ou 1

byte) é separado dos outros por ‘pontos’. Assim, um endereço IP original, descrito em notação binária, como *11000000 10101000 00000000 00000001*, passa a ser representado por *192.168.0.1*. Por convenção, endereços com todos os *bits* de identificação de host iguais à zero, são considerados endereços de identificação da rede, e servem para auxiliar na comutação dos pacotes. Da mesma forma, quando todos esses bits são iguais a um, o endereço é considerado um endereço *broadcast* (difusão). Isso significa que pacotes endereçados a um endereço como este, serão encaminhados a todas as máquinas constituintes da rede da máquina de origem. Nem endereços de identificação de rede, nem endereços de *broadcasting*, podem ser atribuídos aos *hosts* de uma rede [31]. Por exemplo: uma máquina hospedeira com o endereço IP de classe C, *192.168.0.1*, pertence à rede *192.168.0.0*, e *192.168.0.255* é o endereço de difusão dessa rede.

Para evitar confusões e possíveis conflitos entre os endereços de redes privadas e os endereços da Internet, resolveu-se estipular um conjunto de prefixos de endereço reservados, que são utilizados apenas nas configurações das redes proprietárias, e nunca para acesso à Internet. Existe também um endereço de rede não especificado, 0.0.0.0, que não deve ser atribuído nunca a uma interface de rede, e que é utilizado geralmente na configuração de roteadores, para indicar rotas padrões. E há ainda uma faixa de endereços especialmente reservada para *loopback* (*127.0.0/24*), ou seja, utilizada apenas para testar processos de comunicação do padrão TCP/IP em uma mesma máquina local.

Esses endereços especiais pertencem a uma lista maior de endereços reservados para os mais variados fins, e que não podem ser alocados para usuários finais, segundo determinação do IANA [42].

Quando uma instituição deseja montar uma rede local, sem acesso à Internet, ela pode escolher livremente a forma como irá atribuir os endereços IP às suas máquinas, desde que estas sejam organizadas de acordo com os padrões e pertençam a uma mesma rede, ou a redes diferentes que se comuniquem através de roteadores.

No entanto, a internet global de hoje oferece um serviço de comunicação universal, isto é, permite que cada *host* da rede possa se comunicar com qualquer outro computador. Para que seja possível realizar essa comunicação, cada máquina que acessa diretamente a Internet precisa de um endereço IP único. Para garantir essa unicidade dos endereços IP, é preciso a intervenção de algum órgão regulador, responsável pela distribuição dos endereços ou faixas de endereços. Originalmente, esse papel era desempenhado pelo IANA. Hoje, é o ICANN (*Internet Corporation for Assigned Names and Numbers*) quem exerce essa função [7][55]. O mesmo órgão é também responsável pela distribuição de registros de nomes para a Internet e muitos outros serviços, e assim como outras instituições a ele atreladas, é uma organização sem fins lucrativos. O IANA continua sendo o responsável pela administração e controle de novos padrões da Internet.

Na verdade, existem ainda, além do ICANN, outras organizações responsáveis pelas atribuições dos IPs, organizados hierarquicamente. Assim, quando uma instituição decide adquirir endereços para acessar a Internet, ela o faz a um determinado grupo de menor hierarquia, ou seja, mais específico para a sua localidade – um ISP regional, por exemplo. Esse grupo possui endereços já atribuídos por outros grupos maiores, que por sua vez possuem faixas de IP atribuídas por outros órgãos, e assim por diante, até chegar ao ICANN. Existem atualmente cinco instituições maiores de distribuição de endereços, os Registros Regionais de Internet (RIRs), subsidiárias do ICANN, espalhadas pelo mundo.

No Brasil, por exemplo, tal controle é exercido pelo Registro de Endereçamento de Internet para América Latina e Caribe, o LACNIC (*Latin American and Caribbean Internet Addresses Registry*), que distribui IPs tanto para ISPs quanto para usuários finais [71]. A quantidade de endereços é atribuída a uma instituição de acordo com o número de máquinas que irá precisar acessar a Internet.

2.2.3 Aspectos de roteamento

Os datagramas IPv4 são transferidos entre diferentes redes privadas – ou sistemas autônomos – através da comutação de pacotes. A idéia da comutação de pacotes é dividir as mensagens originais a serem transmitidas, em pedaços menores, que são encaminhados a seu destino de forma independente, podendo inclusive seguir caminhos diferentes, através dos roteadores. Diz-se que esse roteamento da Internet é um serviço sem conexão e de melhor esforço: sem conexão, pois as rotas são dinâmicas, podendo ser diferentes para cada pacote, então não há a necessidade de se manter informações de estado dos enlaces em cada roteador; e de melhor esforço, pois não há garantias de preservação da temporização entre pacotes, nem de que os pacotes sejam entregues na mesma ordem em que foram enviados, ou que sejam entregues sem nenhum erro. Nem mesmo a garantia de que os pacotes sejam entregues existe [24].

Existem protocolos específicos para realizar o encaminhamento das mensagens através das redes e sistemas autônomos [25][28][31]. Esses protocolos de roteamento possuem fins específicos e características inerentes às suas tarefas, podendo ser classificados como: intra-SA, ou seja, atua dentro de um mesmo sistema autônomo, como é o caso da maioria dos protocolos de roteamento, como o RIP, o OSPF ou o EIGRP (*Enhanced Interior Gateway Routing Protocol*); ou inter-SA, atuando no roteamento entre diferentes sistemas autônomos, como é o caso do BGP.

Mas, antes de serem roteadas, as mensagens precisam ser empacotadas. Os pacotes ou PDUs da camada de rede, nada mais são do que conjuntos de dados, acompanhados de seus cabeçalhos, que contêm as informações de controle pertinentes à camada. Esses pacotes, por sua vez, são encapsulados em quadros da camada de acesso ao meio – ou camada de enlace, adequadamente aos protocolos e *hardwares* utilizados na rede do *host* de origem, e finalmente transmitidos através dos canais de comunicação (enlaces).

Como os sistemas autônomos possuem suas próprias arquiteturas e padrões de enlace, os roteadores precisam de interfaces de rede que entendam cada um dos meios com os quais for lidar. Um roteador que possua conexões com uma rede Ethernet e com uma rede FDDI (*Fiber Distributed Data Interconnect*), por exemplo, precisa compreender ambos os protocolos, para poder desencapsular os *frames* que chegam por uma das interfaces, extrair informações sobre o endereço destinatário, e novamente encapsular cada *frame* de acordo com o protocolo da interface de destino, repassando-os ao próximo enlace.

Cada enlace possui uma unidade máxima de transferência ou MTU (*Maximum Transfer Unit*), ou seja, uma capacidade de transferência que limita a quantidade máxima de dados que um frame pode carregar por esse enlace [24]. Quando um pacote originalmente encapsulado em um *frame* de maior MTU alcança um roteador que precisa repassá-lo para um enlace de menor MTU, ocorre a fragmentação do pacote, ou seja, o datagrama IP é dividido em pedaços, e encapsulado em *frames* capazes de trafegar pelo enlace de destino. A remontagem desses fragmentos, por motivo de simplificação de protocolo, só é feita no sistema final de destino das mensagens, com a ajuda dos campos de *identificação*, *flag* e *fragmentação* do cabeçalho do datagrama IP: de posse de todos os fragmentos de um mesmo pacote, a camada de rede do destinatário efetua essa remontagem, repassando o pacote remontado para ser desencapsulado pela camada de transporte.

Esse processo de fragmentação de pacotes exige dos roteadores um consumo de processamento e de memória adicionais, que pode retardar um pouco o roteamento dos pacotes, diminuindo o desempenho da rede.

2.3 O protocolo ICMP

O intuito da pilha de protocolos TCP/IP é fazer com que os pacotes sejam encaminhados corretamente, através de roteadores e outros equipamentos de rede, até um último roteador que possua conexão direta com o destino de cada datagrama. Mas, devido a problemas na estrutura da rede, nem sempre isso é possível. Empecilhos como redes congestionadas, falhas de roteamento ou destinos inalcançáveis são comuns de acontecerem.

Para facilitar a tarefa de resolução desses problemas, utiliza-se um protocolo chamado ICMP [7]. Geralmente confundido como parte integrante do protocolo IPv4, o protocolo ICMP é na verdade um protocolo à parte, também pertencente à camada de rede, mas que depende do IPv4, pois é por meio de pacotes IP que as suas informações, assim como a dos demais protocolos da camada de rede e das camadas superiores, são carregadas através da rede. A implementação do protocolo é obrigatória, por isso a confusão acontece.

O ICMP é um mecanismo de gerenciamento e notificação de erros, utilizado pelos roteadores para comunicar ao remetente a ocorrência de possíveis erros durante o transporte dos datagramas. Também é utilizado pelos *hosts* para efetuar testes de comunicação e encontrar fontes de erros. A ferramenta mais conhecida de depuração de erros é o Ping, que utiliza mensagens ICMP do tipo *echo request* e *echo reply* para verificar se um destino pode ser alcançado e qual o tempo de resposta desse destino, atestando o correto funcionamento dos protocolos mais inferiores da pilha TCP/IP.

O ICMP em hipótese alguma resolve problemas, ele apenas comunica à origem a ocorrência de um determinado obstáculo, cabendo portanto ao remetente decidir como tratar o erro, ou encontrar uma possível solução para o impedimento no envio dos pacotes.

2.4 Outros protocolos da camada de rede

Além dos protocolos de roteamento, do ICMP e do IP, a camada de rede do modelo TCP/IP é composta por outros protocolos. Esses protocolos, de complexidades diversas, cumprem funções específicas, e foram sendo acrescentados à medida que novos problemas e aplicações – como o crescimento do tamanho das redes privadas, a transmissão de voz e vídeo em tempo real, ou exigências de segurança e privacidade de dados, surgiam na Internet, dando origem a novos aspectos não levados em conta quando da criação da pilha de protocolos TCP/IP original.

Os protocolos de resolução de endereços ARP (*Address Resolution Protocol*) e de resolução inversa de endereços, RARP (*Reverse Address Resolution Protocol*), por exemplo, servem para mapear os endereços lógicos da camada de rede – endereços IP, para endereços físicos, únicos e reais, da camada de enlace do padrão Ethernet, e vice-versa [7].

O DHCP (*Dynamic Host Configuration Protocol*) [7], que incrementou e substituiu o BOOTP (*Bootstrap Protocol*), é um protocolo de grande utilidade para a administração de grandes redes. Ao invés de atribuir endereços estáticos aos *hosts*, com o DHCP é possível distribuir dinamicamente os endereços IP dentro do domínio de uma rede privada, de forma automática, poupando tempo e problemas como erros de configuração, comumente introduzidos através da configuração manual.

Para permitir a comunicação multicast, realizada entre grupos de computadores, foram introduzidos protocolos de roteamento específicos, além de um protocolo presente nos sistemas finais, o IGMP (*Internet Group Management Protocol*), responsável pela configuração dos grupos, e que age na comunicação entre os *hosts* e os roteadores multicast [24].

Para tentar melhorar o desempenho de aplicações que exigem a demanda de dados em tempo real, como transmissão de vídeo e voz, criou-se o padrão Intserv (*Integrated Services*), que através do protocolo de reserva de largura de banda (RSVP), tenta acrescentar à Internet a garantia da qualidade de serviço (QoS) [24].

Com a popularização da Internet e das redes locais, surgiu também a preocupação das organizações e mesmo de usuários individuais em proteger os dados por eles enviados e recebidos. Um bom exemplo de aplicação que exige tal segurança é o comércio eletrônico. Para permitir um bom nível de segurança na camada de rede, garantindo a privacidade, autenticidade e integridade de dados privados, a solução encontrada foi a adoção de uma estrutura desenvolvida originalmente para o IPv6, conhecida como IPSEC (*IP Secure*), utilizando-a para criar redes virtuais privadas (*Virtual Private Networks* – VPNs) [7].

Devido à introdução, por parte do IETF, desses e de outros protocolos, costuma-se comparar a camada de rede do TCP/IP a uma colcha de retalhos, onde o todo é formado por pedaços frágeis, que vão sendo introduzidos aos poucos, formando um grande remendo.

2.5 Medidas tomadas para evitar o fim dos endereços

Quando o IPv4 foi criado, por volta de 1980, não existia ainda nada parecido com a Internet. O que existiam eram redes proprietárias isoladas, de algumas centenas de *hosts*, cada. Um número de nós insignificante, quando comparado à quantidade de máquinas, equipamentos e redes existentes hoje e conectadas à internet global. Essa expansão foi impulsionada em grande parte pelo surgimento de novas aplicações, que exigiram a criação de mais protocolos e disponibilizaram novos e diferentes serviços, atraindo cada vez mais usuários. Como não havia sido projetado para um número tão grande de usuários, era de se esperar que o IPv4 logo enfrentasse problemas.

Com o passar do tempo, começou a se perceber que da forma como os endereços IP haviam sido criados, não seria possível acomodar a expansão da Internet. Previa-se que em poucos anos haveria uma falta total de novos endereços, impossibilitando a continuidade do crescimento da grande rede. Além disso, as tabelas de roteamento dos roteadores da Internet cresciam descontroladamente, causando problemas de desempenho no acesso à rede. Assim, foi preciso encontrar soluções que pudessem, pelo menos a curto prazo, resolver a escassez dos endereços e frear o crescimento ou mesmo diminuir os tamanhos das tabelas de roteamento. Foram criadas várias novas funcionalidades e novos conceitos, como o NAT e a adoção de sub-redes e super-redes. No entanto, nenhuma delas foi capaz de resolver totalmente o problema, e algumas acabaram até dando origem a novas preocupações.

Das várias soluções implantadas as mais conhecidas são: a utilização de sub-redes; sub-redes de tamanho variável (VLSM); roteamento sem classe entre domínios (CIDR) e *supernetting*; criação de endereços IP privados e tradução de endereços de rede (NAT) [7][25].

2.5.1 Sub-redes

O endereçamento IP original, chamado *classfull*, estava se esgotando rapidamente. Quando uma organização possuía filiais fisicamente distantes umas das outras, ou mesmo setores diferentes que não deveriam pertencer à mesma rede, era necessário adquirir uma nova faixa de endereços IP, o que, com a popularização da Internet, logo daria fim à maioria dos endereços, impossibilitando a expansão da rede.

Essa hierarquia de dois níveis, prefixo de rede e sufixo de *host*, teria que ser modificada, de forma a criar um terceiro nível de hierarquia, interno às organizações, possibilitando a divisão de grandes redes privadas únicas em redes menores, no entanto pertencentes ainda ao mesmo domínio ou sistema autônomo.

A solução adotada foi a criação do conceito de sub-redes. A idéia é tomar alguns dos *bits* originalmente destinados ao endereçamento dos *hosts*, criando assim uma terceira divisão na interpretação do endereço IP. Infelizmente, nessa divisão a primeira e a última sub-redes não eram consideradas válidas, pois poderiam causar problemas no roteamento. Assim, ocorria a perda de uma certa porcentagem de endereçamento. Essa perda, no entanto, valia a pena por possibilitar a subdivisão das redes privadas, e foi abrandada quando depois se passou a permitir a utilização dessas sub-redes antes consideradas inválidas.

Tal divisão em sub-redes deveria ser visível apenas internamente à organização. Para o resto da Internet, apenas o endereço IP original seria utilizado para guiar pacotes até a organização, e roteadores da organização é quem deveriam decidir a qual sub-rede repassar esses pacotes. Essa tarefa era feita justamente com a ajuda da máscara de sub-rede.

Os IPs, que antes eram representados apenas pelo endereço, passaram a ser identificadas também por uma máscara de sub-rede, também no formato de representação decimal separado por pontos. Essa máscara é utilizada pelos roteadores para decidir se deve ou não encaminhar um pacote. Assim, um endereço de classe C, como *192.168.0.2* receberia ainda uma máscara de sub-rede, *255.255.255.0*, indicando que apenas o último octeto poderia ser manipulado pelos administradores da rede local.

As vantagens trazidas pela utilização de máscaras de sub-rede, portanto, foram a possibilidade de organização física das redes privadas, com flexibilidade e de forma transparente ao restante da Internet, evitando o crescimento das tabelas de roteamento externas à organização e diminuindo a taxa de requisição de novas faixas de endereços, prolongando a utilização do IPv4.

2.5.2 VLSM

A máscara de sub-rede de tamanho variável (*Variable-Length Subnet Mask*), VLSM, surgiu para complementar a idéia original da divisão de uma única rede em sub-redes integrantes.

Na idéia original de sub-redes, a divisão do endereço IP original era feita de forma uniforme, isto é, cada sub-rede possuía o mesmo limite de máquinas. O problema era que um setor do órgão poderia ser grande, enquanto outros poderiam possuir uma menor quantidade de máquinas. Ou poderiam existir mais setores do que a divisão em sub-redes de único nível poderia permitir. A divisão em sub-redes original era, portanto, ineficiente, pois fazia com que fossem desperdiçados vários endereços de máquina.

O VLSM consiste em permitir vários níveis de subdivisão da rede, ou seja, várias máscaras de sub-rede de tamanhos diferentes, flexibilizando ainda mais a estruturação da hierarquia interna das empresas. Para permitir sua utilização, foi necessária a criação de novos protocolos de roteamento.

Justamente na época da criação do conceito de VLSM começava a crescer a preocupação com a possível falta de endereços IP, e foi no mesmo período que resolveu se passar a utilizar as sub-redes antes consideradas sem validade (a primeira e a última), otimizando ainda mais o endereçamento de sub-redes.

O VLSM também introduziu uma nova forma de representação da máscara de sub-rede. Ao invés de representá-la com a notação decimal separada por pontos, passou a se acrescentar, ao final do endereço IP, uma barra (/) seguida pela quantidade de bits reservados para identificação

da rede e da sub-rede. Assim, um endereço *192.168.0.2* com máscara *255.255.255.0*, poderia ser agora representado por *192.168.0.2/24*.

Deve existir grande cuidado durante o planejamento das sub-redes, já que um mau planejamento poderia causar ambigüidade na interpretação dos endereços da rede, impossibilitando o seu correto funcionamento.

2.5.3 CIDR

Ambas as soluções anteriores diminuíram a procura pela aquisição de novos endereços IP, mas não resolveram totalmente o problema. O endereçamento ainda era baseado nas três classes cheias comerciais, A, B e C. Se uma empresa possuía mais de 254 máquinas, ou não as possuía ainda, mas previa a expansão, um endereço de classe C tornava-se insuficiente para acomodar todos os hosts. A maioria das organizações acabava optando então por adquirir endereços de classe B, e algumas poucas recorriam às classes A ou C. A desigualdade provocaria o fim dos endereços de classe B em um curto espaço de tempo. Assim, passou-se a convencer as empresas de grande e médio porte a adquirir vários endereços de classe C, ao invés de um único endereço de classe B. Essa medida ia de encontro à divisão em sub-redes, e ainda por cima faria com que as tabelas de roteamento da Internet crescessem de forma descomunal, afetando o desempenho da comunicação através da grande rede.

A solução seria, então, acabar de vez com a noção de classes de endereçamento, de baixa granularidade, através da criação de endereços sem classe (*classless*). Descrita em diversas RFCs, surgiu uma tecnologia que possibilitava um roteamento inter-domínios sem classes (*Classless Inter-Domain Routing*), ou CIDR, cuja idéia principal era adaptar o conceito de sub-redes antes utilizado em redes locais privadas, extendendo-o para ser utilizado em toda a Internet.

Com o CIDR, ao invés de se atribuir às organizações toda uma classe de endereços, passou a se distribuir faixas de IP, suficientes para comportar as redes internas e ainda permitir uma expansão, prolongando ainda mais o tempo de vida dos endereços e do protocolo IPv4. No entanto, o problema do super-crescimento das tabelas de roteamento não estava ainda resolvido. Foi preciso adicionar o conceito de agregação de rotas, criando algo como super-redes (*supernetting*). Com essa abordagem, os roteadores passaram a resumir as suas rotas, transformando blocos de endereços IP adjacentes em uma única entrada em suas tabelas, diminuindo consideravelmente o tamanho das tabelas de roteamento dos roteadores ligados à Internet, e tornando viável a utilização dos endereços *classless*.

A grande desvantagem do CIDR é a sua alta complexidade, se comparado ao roteamento e ao endereçamento de classe cheia originais. Apesar disso, essa é a tecnologia que vem sendo adotada atualmente, juntamente com VLSM e IPs privados e NAT.

2.5.4 IPs privados e NAT

No início da Internet, existiam organizações que desejavam manter apenas uma rede local, pois não viam necessidade em acessar a grande rede. Como não iriam precisar adquirir IPs válidos para a Internet, essas empresas poderiam utilizar qualquer esquema de endereçamento interno, escolhendo trabalhar, por exemplo, com um endereço de classe A ou B, subdividindo-o da forma mais adequada.

Dessa forma, várias redes privadas poderiam estar utilizando esquemas de atribuição de endereços semelhantes. Se uma dessas redes acidentalmente fosse conectada à Internet, poderia ser confundida com uma outra rede, que já possuía acesso à grande rede, causando problemas de roteamento e até mesmo de segurança.

Para evitar esse tipo de conflito, passou a se utilizar alguns blocos de endereço IP privados, ou seja, que não seriam válidos para acesso à Internet, e sim apenas para utilização interna de cada organização. No entanto, se posteriormente essas organizações resolviam finalmente se juntar à Internet, era necessário efetuar a troca de todos os endereços IP de suas máquinas, o que era bastante trabalhoso e geralmente causava uma série de confusões e insatisfações.

Novamente entrou em cena o IETF, que através de suas RFCs, trouxe a público uma solução que resolveria esse e outros problemas, todos de uma vez. O NAT (*Network Address Translation*) consiste em adicionar, a roteadores ou *hosts* específicos, a capacidade de traduzir endereços IP válidos para endereços privados, e vice-versa, através de tabelas de conversão de endereços. Existem quatro tipos de abordagem: NAT unidirecional, NAT bidirecional, NAT baseado em portas e o NAT de sobreposição, sendo o NAT baseado em portas (*Network Address Port Translation* – NAPT) a mais comumente utilizadas hoje em dia. A tabela de tradução também pode ser configurada manualmente.

As vantagens de se utilizar NAT são várias, como: o fim da necessidade de modificar endereços para poder acessar a Internet; compartilhamento de endereços válidos, permitindo a redução de custos de aquisição de IPs e a quantidade de endereços solicitados, ajudando a prolongar a disponibilidade dos endereços IPv4; maior flexibilidade e independência das organizações em relação aos ISPs; maior segurança, já que sem endereços válidos e sem acesso direto à Internet, torna-se mais complicada a invasão de uma rede local.

Mas o NAT também possui desvantagens. Algumas delas são: complexidade na resolução de problemas, devido à tradução de endereços, que gera confusões; incompatibilidade com alguns protocolos; queda de desempenho, já que toda requisição interna de acesso à Internet, bem como a resposta a tal requisição, precisa ser traduzida pelo servidor NAT, gerando um atraso no tempo de resposta e uma carga extra de processamento na rede local; impossibilidade de comunicação fim-a-fim, isto é, comunicação direta entre dois *hosts* pertencentes a redes locais diferentes [7].

Capítulo 3

O protocolo IPv6

Nenhuma das medidas apresentadas no Capítulo 2 foi capaz de tranqüilizar os órgãos de distribuição de endereços e demais mantenedores da Internet. O crescimento e popularização da rede continuaram, de forma cada vez maior, principalmente com o advento da *World Wide Web*, e posteriormente de outras aplicações. Na verdade, buscando resolver problemas de endereçamento, criou-se um outro conjunto de problemas. A complexidade de endereçamento e o número de protocolos adicionais aumentaram deveras, sem falar que a adoção do NAT acabou de vez com a comunicação fim-a-fim, impossibilitando a troca direta de dados entre usuários de redes distintas. Outro problema é a diminuição da capacidade efetiva de endereçamento, devido às faixas de endereço reservadas para utilizações especiais.

Já que as soluções existentes não serão capazes de evitar o fim dos endereços IP versão 4, foi preciso desenvolver uma nova alternativa. A resposta encontrada pelos pesquisadores, foi criar um novo protocolo de internet: o IP *Next Generation* (IPng), que aborda e engloba inclusive questões que não haviam sido levadas em consideração no desenvolvimento do IPv4.

No início da década de 90, já se previa a possível estagnação, em um futuro próximo, da recém criada Internet, devido à escassez dos endereços IP de 32 *bits*. Uma grande quantidade de novos protocolos foi proposta por vários pesquisadores de todo o mundo. Entre 1991 e 1993 foram submetidos aos órgãos reguladores várias RFCs, restando três candidatos principais. Uma das propostas finalistas, o *Simple* IP (SIP), foi aprimorada e mesclada às idéias de outras propostas, e em meados de 1994 deu origem a uma primeira proposta de esboço (*proposed draft*) oficial do IETF para um novo modelo de endereçamento. Algum tempo depois, a proposta passou a ser considerada um esboço padrão (*standard draft*), e logo em seguida seria adotada como padrão do novo protocolo, dando origem a uma série de RFCs, e passando a ser chamada de IPv6.

Essa nova versão do protocolo IP surgiu primordialmente como solução para evitar a iminente falta de endereços IPv4, mas acabou incorporando muitas outras melhorias e funcionalidades. A fragmentação de pacotes durante o trajeto, por exemplo, foi totalmente abolida nessa nova versão, por tomar um bom tempo de processamento.

A futura utilização do IPv6 já é um fato confirmado pelos órgãos de controle da Internet, mas não se tem exata certeza de quando será totalmente implantada, já que a possível falta de endereços IPv4 foi abrandada, com a adoção do NAT, do CIDR e das sub-redes. Algumas das principais características desse novo protocolo são listadas a seguir [18].

Auto-configuração: um mecanismo que torna a migração, administração e a implantação de novos dispositivos de rede mais fáceis e menos trabalhosas.

Conectividade fim-a-fim: como a quantidade de endereços possíveis é muito grande, tornou-se desnecessária a utilização de tecnologias como o NAT e as máscaras de sub-rede, permitindo assim que cada *host* esteja realmente conectado à Internet através de endereços reais válidos.

Endereçamento de 128 bits: como o intuito principal do IETF era evitar a escassez de endereços, foi criado esse novo formato de endereçamento, onde os *bits* são divididos em grupos de 16 e descritos no formato hexadecimal separado por 'dois pontos'. Assim, é capaz de atribuir teoricamente cerca de $3{,}4x10^{38}$ diferentes endereços, uma quantidade bem maior do que os $4{,}3x10^{9}$ oferecidos através do IPv4 [30].

Extensibilidade: Através da utilização de *Cabeçalhos de Extensão*, determinados pacotes IPv6 podem conter informações adicionais às informações comuns, permitindo, por exemplo, a fragmentação, que, por padrão, no novo protocolo não acontece.

Mobilidade com eficiência: existe ainda muita pesquisa quanto à utilização de dispositivos móveis com redes IPv6, principalmente no que diz respeito à segurança. Mas o novo protocolo possui uma série de vantagens, em relação ao IPv4, para lidar com a transparência exigida pela mobilidade.

Rotulação e prioridade de fluxo: existem campos no cabeçalho IPv6 específicos para classificar os tipos de pacotes, permitindo lidar de forma muito mais simplificada com questões de Qualidade de Serviço (QoS).

Segurança incorporada: utilização opcional de algoritmos de criptografia DES/3DES para cifrar as mensagens durante todo o trajeto dos pacotes.

Simplificação do cabeçalho: o grupo responsável pela implementação do IPv6 verificou que alguns campos e funções do protocolo IPv4 executavam tarefas que não eram necessárias, tornando o trabalho do protocolo lento. Alguns campos foram removidos, outros renomeados e movidos de lugar e um outro adicionado. Além disso, o cabeçalho que no IPv4 era de tamanho variável, no IPv6 passou a um tamanho fixo de 40 *bytes*.

De acordo com o surgimento de novos dispositivos que necessitam de uma conexão IP para maior aplicabilidade e liberdade, reforça-se cada vez mais como tendência o IPv6, e estima-se que em menos de 15 anos a migração completa já terá acontecido [14]. A intenção é que o IPv6 sirva como alicerce para a integração entre vários tipos de serviços, como redes de telefonia móvel e fixa, e permita o ingresso à rede de novos tipos de aplicações, como a televisão interativa, aparelhos domésticos com acesso à Internet, ou até mesmo a computação ubíqua ou pervarsiva [14][30][33][38][41].

Por isso, o IPv6 já está em fase de utilização em muitos países da Europa e da Ásia, disponibilizado através de órgãos de pesquisa como o 6BONE, que possui uma rede IPv6 experimental a fim de examinar o comportamento dessa tecnologia [43]. A rede do 6BONE tem crescido bastante, interligando-se a outras redes IPv6, e já começa a ser conhecida como o *backbone* da Internet IPv6.

Existe um número cada vez maior de empresas privadas e organizações realizando pesquisas relacionadas ao IPv6 [59]. Muitos governos, como os do Japão e o da Suécia, estão

dando incentivos fiscais, como a isenção de impostos de 100% aos produtos que forem produzidos em seus paises e que já estejam prontos para o novo padrão do IP [61]. Nas Américas, o ARIN (*American Registry Internet Number*), o LACNIC e o LACIPv6TF (*Latin American and Caribbean Task Force IPv6*) são as instituições responsáveis pela administração das redes continentais, e já estão prontas para a venda dos blocos de endereços IPv6 para *links* de Internet [15][34][45][70][71].

No Brasil, o BR6BONE [47], atrelado à Rede Nacional de Pesquisas (RNP) [81] e filiado ao 6BONE europeu desde 1998, é uma instituição governamental que estuda e analisa o IPv6 para fins de pesquisas e aplicações governamentais. Além disso, vários grupos de trabalhos estão sendo formados para estudar as aplicações possíveis para o IPv6 no Brasil, e já existem empresas privadas especializadas no assunto, oferecendo treinamentos e serviços de migração e implantação do novo protocolo em organizações.

3.1 O cabeçalho de pacotes IPv6

O cabeçalho do datagrama IPv6 sofreu várias alterações e simplificações em relação ao cabeçalho IPv4, objetivando maior velocidade e melhor desempenho na comunicação, como pode ser visto na Figura 3.

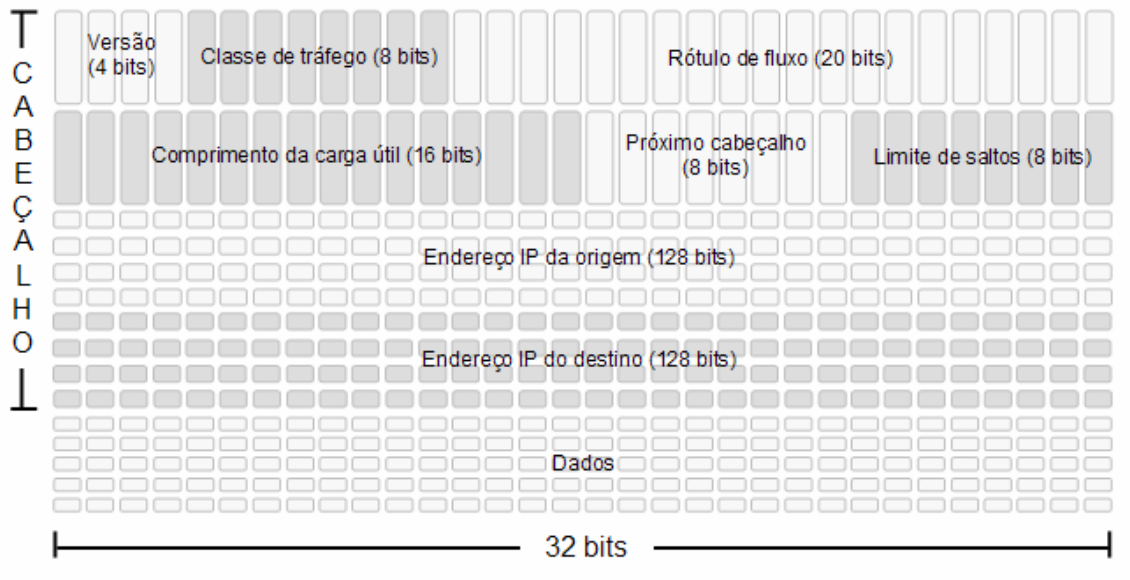

Figura 3. Os campos do datagrama IPv6.

A seguir, novamente baseando-se em Kurose [24], descreve-se cada campo do datagrama IPv6, de forma condensada.

Versão: da mesma forma que o campo equivalente no cabeçalho IPv4, serve para indicar a versão do protocolo IP utilizada pelo datagrama.

Classe de tráfego: tem função semelhante ao campo *tipo de serviço* do datagrama IPv4.

Rótulo de fluxo: é utilizado para identificar fluxos de dados, um conceito não muito bem definido, mas que está relacionado a questões de qualidade de serviço e ao tratamento de informações que possuam maior prioridade.

Comprimento da carga útil: indica o comprimento do datagrama IP. Não é preciso um campo para informar o comprimento do cabeçalho, que nessa versão do IP possui um tamanho fixo de 40 *bytes*.

Próximo cabeçalho: equivale ao campo *protocolo da camada superior* do IPv4, utilizando inclusive os mesmos valores.

Limite de saltos: equivalente ao campo *tempo de vida* do cabeçalho IPv4.

Endereços IP da origem e do destino: carregam, respectivamente, os endereços de 128 *bits* do remetente e do destinatário.

Dados: contém os dados das camadas superiores e da mensagem propriamente dita.

3.2 O formato do endereçamento IPv6

Enquanto o IP versão 4 utiliza endereços com 32 *bits* de comprimento, o IPv6 introduz um novo esquema de endereçamento, com comprimento de 128 *bits*. Esse novo endereço exige uma representação diferente da representação decimal separada por pontos, utilizada como IPv4.

A notação utilizada com o IPv6 é a hexadecimal separada por ‘dois pontos’ [2][20]. O bloco de 16 *bytes* é subdividido em 8 grupos de 16 *bits* cada. Estes grupos são descritos em formato hexadecimal, e separados uns dos outros por ‘dois pontos’. Um exemplo de endereço IPv6 válido é *3ffe:3102:0000:0000:0008:0800:200C:417A*. Essa notação foi escolhida por ser mais compacta do que a notação decimal separada por pontos.

Para facilitar a compreensão do endereço, abreviações podem ser utilizadas. Grupos formados por zero podem ser descritos como um único zero. O mesmo é válido para zeros sobrando à esquerda, em um grupo. Com essas abreviações, o endereço anterior ficaria assim: *3ffe:3102:0:0:8:800:200C:417A*. Há ainda um outro tipo de abreviação que pode ser utilizado. Vários grupos de zero, seguidos, podem ser substituídos por ‘dois pontos’ seguidos por ‘dois pontos’ novamente. Essa abreviação só pode ser utilizada uma vez no endereço, pois se utilizado mais de uma vez, ficaria impossível descobrir a quantidade de *bits* abreviada. Novamente utilizando o endereço supracitado, ele ficaria assim: *3ffe:3102::8:800:200C:417A*.

Essas abreviações serão muito úteis na fase de início de utilização do protocolo, pois a quantidade de endereços alocados é e continuará sendo, ainda por um bom tempo, muito pequena. Hoje, apenas cerca de 15% dos endereços IPv6 já foram reservados para futura alocação [18]. Isso deixa espaço para a utilização dessas abreviações. No entanto, deve-se tomar cuidado com a utilização desses endereços comprimidos. Por exemplo, a forma correta de abreviar o prefixo do endereço *CAFF:CA01:0000:0056:0000:ABCD:EF12:1234/64* seria *CAFF:CA01:0:56::/64*, e não *CAFF:CA01::56/64*, evitando assim a ocorrência de ambigüidade na interpretação do endereço estendido [18].

Em contraste à máscara de sub-rede e à notação CIDR, utilizadas com o IPv4, o IPv6 também apresenta um conceito de prefixo, para indicar a parte de um endereço dedicada à identificação da rede. Um exemplo de notação de prefixo é *3ffe:3102::8:800:200C:417A*/16. A barra seguida por um número, neste caso 16, indica a quantidade de *bits* mais à esquerda utilizados para identificar o tipo do endereço IPv6. Existe uma lista de formatos de endereços já reservados pelo IANA, chamados de prefixo de formato, ou prefixo de roteamento global, como pode ser visto na Tabela 1. O prefixo de formato é parte integrante do prefixo de identificação de um endereço, e não deve ser confundido com o mesmo.

Tabela 1. Lista de prefixos reservados pelo IANA.

Alocação	Prefixo binário	Prefixo hexadecimal
Reservado	0000 0000	::0/128
Reservado para alocação NSAP	0000 001	
Reservado para alocação IPX (removido no RFC mais recente)	0000 010	
Endereço unicast Global	001	
Endereço unicast Link-local	1111 1110 10	FE80::/10
Endereço unicast Site-local	1111 1110 11	FEC0::/10
Endereço multicast	1111 1111	FF00::/8

Fonte: HAGEM, SILVIA. **IPv6 Essentials**. California: O'Reilly & Associates, 2002. p. 31.

Essa lista não é imutável. Endereços inicialmente reservados para um fim específico podem ser posteriormente re-alocados com outras finalidades. Essas mudanças, no entanto, só devem ser comuns na fase inicial de migração para o IPv6, enquanto o protocolo não se consolida.

No IP versão 4, os endereços são classificados, quanto à transmissão de dados, como *unicast* e *broadcast*. O *unicast* é um endereço único, que serve para identificar uma única interface de rede. O *broadcast* é um endereço especial, utilizado para enviar mensagens para os computadores pertencentes a toda uma rede local.

O IPv6 aboliu o uso de endereços de *broadcast*. Em seu lugar, entram os endereços *multicast*. Um endereço multicast serve para identificar todo um grupo de interfaces. Inclusive, o *broadcast* pode ser encarado como um caso particular de *multicast* [24]. O endereço *multicast* já vinha sendo utilizado com o IPv4, mas era preciso algum investimento para preparar a rede para tal comunicação, como troca ou *upgrade* de alguns equipamentos, e adoção de protocolos específicos, como o IGMP, enquanto no IPv6 o suporte ao roteamento *multicast* é intrínseco [18].

O IPv6 adotou também um novo tipo de endereço, especificado através da RFC 1546, o endereço *anycast* [57]. Um único endereço *anycast* pode ser atribuído a várias interfaces de rede simultaneamente. Assim, o computador que estiver mais próximo à origem de uma requisição *anycast*, é quem responde a mensagem.

Uma máquina configurada para comunicação IPv6 geralmente possui mais de um endereço por interface [2][18][20]. No mínimo, cada interface possui um endereço de escopo de *link*, seja ele do tipo *link-local* ou *site-local*. Pode possuir ainda quantos endereços *unicast* forem necessários, e alguns endereços *multicast*, se fizer parte de algum grupo *multicast*. Se um *host* pretende obter acesso à Internet IPv6, por exemplo, deve possuir, além do endereço de escopo de *link*, um endereço *unicast* do tipo global.

Endereços de escopo de *link* são endereços obtidos através de auto-configuração, ou via DHCPv6. Esse tipo de endereço jamais deve ser roteado para a Internet. Ele apenas é utilizado para a comunicação com computadores diretamente conectados, pertencentes a uma mesma rede local. Esses computadores são chamados de vizinhos. A diferença entre os tipos *link-local* e *site-local*, é que o segundo tipo possui uma informação a mais do que o primeiro, que diz respeito à sub-redes, podendo ser roteado internamente, e tornando possível a divisão lógica da rede local. A figura a seguir mostra os dois tipos de endereço. Os 10 *bits* mais significativos constituem o prefixo global reservado para a identificação do tipo do endereço, descrito na tabela anterior, e juntos com os próximos 54 *bits*, formam o prefixo de rede do endereço.

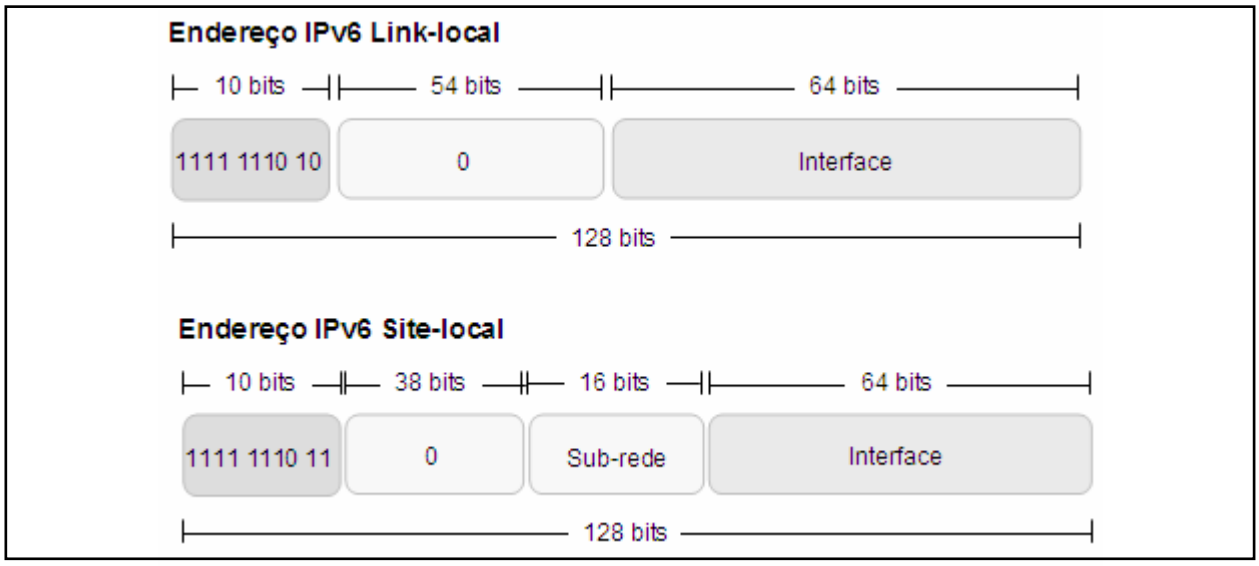

Figura 4. Formato de endereços de escopo de *link*.
Fonte: HAGEM, SILVIA. **IPv6 Essentials**. California: O'Reilly & Associates, 2002. p. 34.

Já o endereço do tipo global, é aquele utilizado para comunicação através dos roteadores, podendo percorrer os vários sistemas autônomos espalhados pela Internet [42]. O formato desse endereço é mostrado na Figura 5. Os três primeiros *bits* são o prefixo de formato. O campo TLA (*top-level agregation identifier*) contém as informações de determinação de rotas de mais alto nível, e sua função principal é a otimização do roteamento. O campo RES está reservado para uso futuro. O campo NLA (*next-level agregation identifier*) é o pedaço de endereço distribuído aos provedores de serviço, como o LACNIC ou o ARIN, que por sua vez utilizam o campo SLA (*site-level agregation identifier*) para identificar cada rede privada, ou assinante, que se liga a eles. O campo *Interface* é montado a partir dos 48 *bits* do endereço MAC (*Medium Access Protocol*), se a rede for Ethernet, ou através de um processo equivalente para outras tecnologias como o FDDI, por exemplo [18].

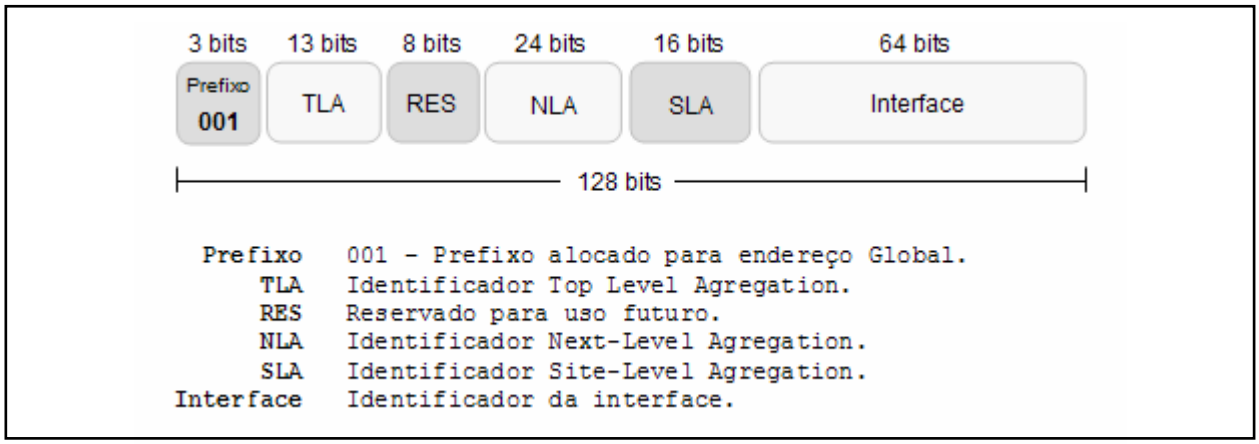

Figura 5. Formato de endereço IPv6 Global.
Fonte: HAGEM, SILVIA. **IPv6 Essentials**. California: O'Reilly & Associates, 2002. p. 35.

Existem ainda alguns endereços IPv6 considerados especiais. O endereço *0:0:0:0:0:0:0:0* é chamado de endereço não-especificado, e só é utilizado durante o processo de inicialização de uma máquina, antes de obter um endereço válido. O IPv6 também possui um endereço de loopback, o *0:0:0:0:0:0:0:1*, que pode ser abreviado para *::1*. Existem ainda dois endereços IPv6

com IPv4 embutido. A diferença entre os dois é quase nenhuma, como pode ser visto na Figura 6. Ambos preenchem os últimos 32 *bits*, os de menor ordem, com o endereço IPv4 da interface. O endereço *IPv4-compatível* é utilizado em conexões de tunelamento automático, enquanto o endereço *IPv4-mapeado* pode ser utilizado para a comunicação dos *hosts* IPv6 com *hosts* IPv4 [18]. Existe ainda um outro endereço especial, utilizado para a comunicação de túneis *6to4*, discutidos mais adiante.

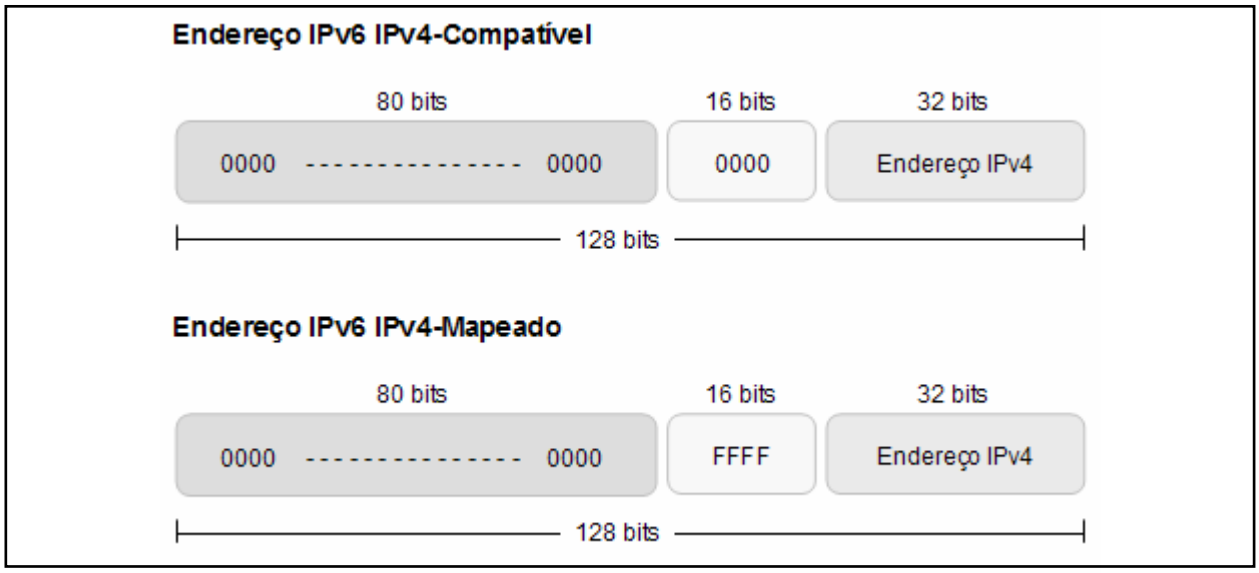

Figura 6. Formato de endereços IPv6 com IPv4 embutido.
Fonte: HAGEM, SILVIA. **IPv6 Essentials**. California: O'Reilly & Associates, 2002. p. 38.

Outro ponto importante a ser discutido, no que diz respeito ao endereçamento IPv6, é a capacidade de auto-configuração dos componentes da rede. É um processo novo, e uma característica chave quando for necessária a integração à rede IPv6, de aparelhos como eletrodomésticos, dispositivos móveis, dentre outros.

Quando não há roteadores na rede, as máquinas se auto-atribuem endereços IPv6 *unicast* de escopo de *link*, obtidos com base na especificação *Extended Unique Interface* (EUI-64) [7]. Esse endereço é construído a partir dos 48 *bits* do endereço MAC da interface, aos quais são acrescentados mais 16 *bits*, sendo, portanto, de identificação única.

Se existir algum roteador na rede, é ele quem irá distribuir os endereços das interfaces dos computadores da rede local. Esse processo é feito através de uma simples configuração, predefinindo a faixa de endereços a ser atribuída. A partir daí, sempre que um *host* se conecta à rede, através de mensagens ICMPv6 de descobrimento de vizinhos e de anúncio de roteador, este atribui a cada interface da rede um endereço pertencente à faixa predeterminada.

O IPv6 também suporta DHCP, agora chamado de DHCPv6, e que funciona de forma semelhante ao seu predecessor. A auto-configuração e o DHCPv6 podem, inclusive serem combinados em uma mesma rede. No entanto, a auto-configuração é vantajosa em relação ao DHCPv6, pois permite uma maior flexibilidade quanto a mudanças de endereço, e um esforço menor de configuração dos elementos da rede.

A auto-configuração é chamada de configuração sem estado (*stateless*), enquanto a utilização do DHCPv6 é apelidada de configuração com estado (*statefull*) [37].

3.3 Aspectos de roteamento

O roteamento de pacotes IPv6 possui algumas diferenças em relação ao roteamento IPv4. Essas mudanças – que têm como objetivo otimizar o desempenho do processo de comutação dos pacotes, exigem protocolos de roteamento que suportem as novas tarefas, ou pelo menos que adaptações nos protocolos atuais sejam implementadas.

Diversos protocolos de roteamento já suportam o IPv6: o RIPng (RIP *next generation*), desenvolvido a partir dos protocolos RIPv1 e RIPv2; o OSPFv3, que mantém as características do OSPF para IPv4, acrescentando apenas algumas mudanças necessárias para adaptação ao IPv6; o BGP-4, que passa a lidar com o novo protocolo através de algumas extensões adicionais. Existem ainda outros protocolos de roteamento sendo adaptados para trabalhar com o IPv6 [18].

Uma das alterações mais importantes em relação ao roteamento de pacotes no protocolo IPv6 diz respeito à fragmentação. Como já foi citado, se um roteador IPv4 encontra um enlace de destino com menor MTU do que o da origem, precisa fragmentar o pacote para mandar adiante. Os projetistas do IPv6 notaram que esse processo de fragmentação de pacotes era ineficiente, por ocupar muito tempo de processamento, atrasando a comutação dos pacotes.

Assim, por padrão, o roteamento IPv6 não permite a fragmentação de pacotes durante o caminho, ou seja, realizada por roteadores. Agora, o único responsável pela fragmentação é o remetente das mensagens. Como não há mais a fragmentação nos roteadores, o cabeçalho do IPv6 aboliu os campos de controle dessa operação que existiam no IPv4: *identificação*, *flag* e *fragmentação*.

Com o IPv6, quando um *host* deseja enviar uma mensagem a algum destinatário, assume que o MTU de destino é o mesmo de sua rede local. Se durante o caminho algum roteador verifica a existência de um MTU menor, o pacote é descartado e uma mensagem ICMPv6 do tipo *pacote grande demais* (*packet too big*) é retornada para o remetente, contendo o novo tamanho indicado de MTU. Assim, o remetente passa agora a encapsular os pacotes de acordo com esse novo valor. Durante o caminho, MTUs menores podem ser encontrados novamente, repetindo o processo. Esse processo de descoberta de MTU é chamado de *Path MTU Discovery*, e só se dá por encerrado quando os pacotes finalmente atingem o seu destino.

É importante frisar que a fragmentação de pacotes IPv6 realizada por roteadores pode ser necessária, por motivos diversos. Sendo assim, apesar de não ser adotada por padrão, e de não ser indicada, essa fragmentação pode sim ser realizada, através do uso de um cabeçalho de extensão, indicado no campo *próximo cabeçalho* do pacote IPv6 [42].

3.4 ICMPv6

Relembrando o que foi dito na *seção 2.3*, o protocolo ICMP é um importante protocolo da camada de rede, dependente do protocolo IP, e que auxilia na resolução de possíveis problemas de comunicação em uma rede. É um mecanismo de gerenciamento e notificação do estado de uma rede.

O ICMPv6 é a versão do protocolo ICMP utilizada com o IPv6. Apesar de servir para os mesmos fins e de possuir estrutura de pacote parecida com a de seu predecessor, o ICMPv6 é muito mais elaborado, mais poderoso. Isso porque toda sorte de protocolos auxiliares presentes na camada de rede, como o DHCP, o ARP/RARP e o IGMP, foram englobados pelo ICMPv6 [18]. E, além disso, novas funcionalidades, como a descoberta de vizinhos (*Neighbor Discovery* –

ND) e a descoberta de MTU (*Path MTU Discovery*), foram implementadas. Devido a essas medidas, fica evidente a falta de compatibilidade entre as versões do ICMP.

Basicamente, o processo de descoberta de vizinhos substitui, com melhor desempenho, os protocolos ARP e RARP e agrega novas funções. Através de tipos específicos de mensagens ICMPv6, o ND encontra *hosts* e roteadores conectados em um mesmo *link* local, determina os vizinhos que estão acessíveis em um dado momento, e verifica eventuais mudanças de endereços desses vizinhos. É ele também quem possibilita a existência do processo de auto-configuração, sem estado, do IPv6.

O protocolo de gerência de grupos *multicast*, IGMP, também foi transformado em mensagens ICMPv6, onde passou a se chamar MLD (*Multicast Listener Discovery*) [18].

Questões relativas à segurança e prevenção de certos tipos de ataques, foram previstas para o ICMPv6 [18]: muitas das mensagens possuem o valor máximo de 255 para o campo *hop limit* do cabeçalho do pacote ICMP, evitando respostas a pedidos com valores inferiores, e que *hosts* remotos se intrometam na rede; para proteção contra ataques de negação de serviço, utiliza-se uma função de limite de taxa de mensagens ICMPv6, que pode ser baseada em larguras de banda ou em intervalos de tempo.

3.5 Camada de enlace

Como era de se esperar, o IPv6 é capaz de trabalhar baseado em todos os padrões da camada de enlace que já eram utilizados com o IPv4.

Quando há necessidade de auto-configuração em uma rede Ethernet, o IPv6 utiliza uma especificação chamada de EUI-64 para, baseado no endereço MAC da interface, montar um endereço IPv6 de escopo de *link*, e de identificação única. O processo consiste em converter os 48 *bits* do endereço MAC em 64 *bits*, utilizando-o para preencher o campo *interface* do endereço IPv6.

Para todos os outros padrões da camada de enlace, como o *Token Ring*, o FDDI, o *Frame Relay*, ou o ATM (*Assynchronous Transfer Mode*), existe também algum processo equivalente, mantendo a capacidade do protocolo IP de integrar diferentes tecnologias em uma mesma única rede [18]. Até mesmo o IPX (*Internetwork Packet Exchange*), utilizado em redes Novell e prestes a ser extinto, é suportado pelo IPv6.

3.6 Protocolos das camadas superiores

A pilha de protocolos TCP/IP foi projetada em camadas, de maneira que quaisquer alterações que precisassem ser implantadas em certa camada tivessem o seu efeito de propagação, pelas demais camadas, minimizado.

Sendo assim, o impacto causado nas camadas mais superiores na pilha de protocolos, pela mudança da versão do protocolo IP consiste basicamente em realizar pequenas mudanças, para que protocolos como UDP, TCP, DHCP, FTP (*File Transfer Protocol*), Telnet, HTTP (*HyperText Tranfer Protocol*), dentre outros, passem a reconhecer o novo formato de endereçamento de 128 *bits* do IPv6 [18].

O protocolo mais afetado pela migração do protocolo IP foi o DNS (*Domain Name System*) [5]. Um novo conceito, de DNS Dinâmico, foi introduzido, e novos tipos de registro foram criados: o AAAA ou quad-A, e o A6. Também mecanismos de segurança, conhecidos como *Domain Name System Security Extensions* (DNSSEC), foram apresentados [1].

Capítulo 4

Migração e interoperabilidade

São muitas as vantagens prometidas pelos entusiastas do novo protocolo da Internet, mas como só agora têm surgido os primeiros *backbones* e redes IPv6 comerciais, os benefícios reais de tal protocolo são ainda praticamente desconhecidos, à exceção do aumento da quantidade de possíveis endereços, desmerecendo o novo protocolo perante a comunidade, e causando mais adiamentos na migração das versões do protocolo IP.

Além disso, os custos envolvidos com a transição entre os protocolos são muitos, e elevados. Seria totalmente inviável estabelecer, por exemplo, uma data e um horário para que todas as máquinas da Internet fossem desligadas e atualizadas, devido à enorme quantidade de máquinas e gerenciadores de redes envolvidos em tal processo. A implantação da nova pilha de protocolos IPv6 poderia ocorrer rapidamente em alguns casos, enquanto outros poderiam encontrar problemas durante o *upgrade*, demorando demasiadamente a efetuar a transição. O transtorno causado por problemas assim seria imenso e poderia deixar a Internet desativada por um longo intervalo de tempo. Além disso, equipamentos de *hardware* teriam que ser completamente – ou quase completamente – substituídos, exigindo grandes investimentos por parte das instituições – universidades, provedores de acesso à Internet, empresas, etc. – e causando desperdício desses equipamentos, que poderiam ainda ser utilizados por certo período, até que se tornassem obsoletos.

Portanto, o maior empecilho na implantação do IPv6 é que as mudanças introduzidas pela nova versão causam algumas incompatibilidades com a versão atualmente utilizada [37]. Assim, existe a necessidade da coexistência de ambas as versões do protocolo por um bom período de tempo, evitando, por exemplo, o emprego de investimentos demasiadamente onerosos. Faz-se necessária a adoção de propostas de transição gradativas, a serem realizadas em longo prazo.

Existem hoje várias abordagens de migração lenta [5][18][37], sendo as mais conhecidas a de pilha dupla e o tunelamento. Uma outra abordagem existente é a tradução de endereços. Cada método tem seus pontos fortes e fracos, e aplicabilidades diferentes. Devem então, sempre que possível, serem combinados para permitir a maior acessibilidade IPv6 possível para uma rede.

4.1 Pilha Dupla

A Pilha Dupla é a maneira mais direta de se manter a compatibilidade entre os protocolos. Consiste em se utilizar equipamentos que possuam a capacidade de se comunicar tanto através do IPv4 como do IPv6. Quando estiver inter-operando com um outro equipamento que suporte apenas IPv4, o equipamento IPv6/IPv4 utilizará datagramas IPv4, e quando for possível interagir com nós que possuam suporte ao IPv6, utilizará datagramas IPv6. A distinção entre qual versão do protocolo utilizar é feita através do DNS, que devolve um endereço IPv6, se o nome do *host* de destino a ser resolvido for habilitado a IPv6, ou devolve um endereço IPv4 em caso contrário.

O problema dessa abordagem é que dois sistemas finais habilitados para lidar com o IPv6 podem acabar se comunicando através de IPv4, se a única forma de se chegar do remetente ao destinatário for passando por pelo menos um equipamento que lide apenas com IPv4. Assim, as informações de cabeçalho contidas no IPv6 original serão perdidas, e os *hosts*, que poderiam fazer uso do IPv6, não poderão usufruir de suas vantagens, não obtendo, portanto, o melhor desempenho desejado [6].

Outra questão a se levar em conta, é que máquinas que trabalhem com ambos os protocolos ao mesmo tempo podem facilmente ser sobrecarregadas: um roteador, por exemplo, deverá possuir tabelas de roteamento distintas para cada protocolo; também os comandos para configurar cada protocolo são diferentes. Sem contar que para trabalhar com ambas as versões do protocolo ao mesmo tempo, o processamento da máquina deverá ser praticamente dobrado.

Por fim, ainda são poucos os provedores de serviços que possuem estruturas com suporte ao protocolo IPv6, e que podem, por conseguinte, disponibilizar conexões IPv6, dificultando a utilização dessa abordagem, e praticamente restringindo-a a âmbitos de pesquisa.

4.2 Tradução de endereços (NAT-PT)

Este método é uma alternativa à utilização de pilha dupla. Ao invés de trabalhar com ambos os protocolos, as máquinas de um seguimento de rede IPv6 podem ser configuradas para funcionar apenas com IPv6. Quando for necessário efetuar comunicação com *hosts* IPv4, essas máquinas utilizarão um *gateway*, com capacidade de efetuar a tradução de mensagens IPv6 em IPv4 e ICMPv6 em ICMP, e vice-versa. Para que essa tradução possa ocorrer, a máquina *gateway* precisa mapear os campos de uma versão do IP para a outra.

A utilização desse método de acesso, uma espécie de NAT específico para conversão de versões dos pacotes IP, conhecido como NAT-PT (*Network Address Translation and Protocol Translation*), deve ser evitada [18]. Nem todos os campos dos cabeçalhos são equivalentes ou passíveis de tradução, principalmente quando há a utilização de opções específicas para cada versão. Essa não totalidade de tradução entre as versões acarreta na incapacidade de utilização de todos os recursos oferecidos pelo IPv6, quando necessária a comunicação com máquinas IPv4.

Na verdade, aconselha-se a utilização dessa forma de obtenção de acesso à rede IPv6 apenas num primeiro momento da migração de uma rede, e somente se não for possível a utilização de pilha dupla ou o tunelamento.

Outra desvantagem do método é que máquinas comunicando-se com versões diferentes do protocolo IP não conseguem estabelecer uma conversação fim-a-fim segura através do IPSEC, por questões de incompatibilidade do *framework*, que funciona diferente em cada uma das versões do IP.

4.3 Tunelamento

O tunelamento, assim como a pilha dupla, exige da máquina o suporte para ambas as versões do protocolo IP. No entanto, a forma como os protocolos interagem entre si é particularmente distinta.

A idéia básica dos túneis é encapsular pacotes IPv6 em pacotes IPv4, permitindo que sejam encaminhados através da infra-estrutura já existente da Internet, até que cada pacote chegue ao seu destino, onde será desencapsulado e interpretado novamente como IPv6 (Figura 7). Os equipamentos intermediários não precisam saber que o conteúdo do pacote IP que estão transmitindo é na verdade um pacote IPv6 encapsulado. Se um equipamento com suporte a IPv6 detecta que realizará comunicação com um equipamento IPv4, ele próprio encapsula o datagrama, efetuando a comunicação com o vizinho através de IPv4. O próximo nó IPv6 deverá receber então um pacote IPv4 e desencapsular tal pacote. Dessa forma, a transmissão das mensagens IPv6 ocorre de maneira transparente para os nós IPv4, e possibilita a comunicação entre sistemas finais através do IPv6, utilizando-se de toda a base da Internet IPv4 atual.

O equipamento que encapsula os pacotes é chamado de *ponto de entrada do túnel*, enquanto o equipamento que recebe esses pacotes e efetua o desencapsulamento é chamado de ponto *de saída do túnel*. Tanto o ponto de entrada, quanto o ponto de saída, pode ser um roteador ou uma máquina *host*. Para que o ponto de saída saiba identificar a presença do tunelamento, o campo *protocolo da camada superior*, do pacote IPv4, é configurado como 41.

É interessante notar que todo o caminho percorrido através da estrutura IPv4 é considerado pelas pontas do tunelamento como um único enlace, independente da quantidade de saltos realizados. Assim, ao receber datagramas IPv4 contendo pacotes IPv6 encapsulados, o ponto de saída do túnel desencapsula esses pacotes, e decrementa por um o campo *limite de saltos* do cabeçalho IPv6, já desencapsulado, antes de enviá-lo ao destino final.

Os túneis podem ser implementados de maneira automática, ou através de configuração explícita. Um túnel é dito automático quando não há necessidade de configuração do ponto de saída do túnel, no ponto de entrada. Esse ponto de saída é escolhido dinamicamente. Já em um túnel manual, é preciso indicar previamente, durante a configuração do ponto de entrada do túnel, o endereço que será utilizado como ponto de saída do túnel.

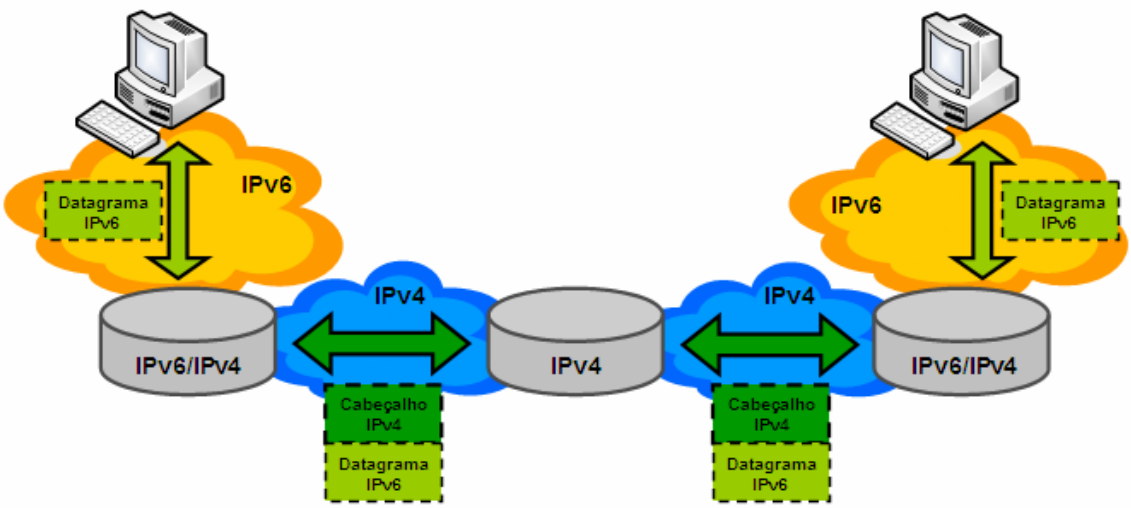

Figura 7. Túnel: os roteadores IPv6/IPv4 encapsulam os pacotes IPv6, transmitindo-os de forma transparente através de redes e equipamentos IPv4. Ao alcançar um outro nó habilitado ao IPv6, o pacote é desencapsulado e encaminhado corretamente ao seu destino através de redes IPv6.

4.3.1 Tunelamento automático

Existem vários tipos de tunelamento automático. O tipo mais comum é o que utiliza endereços IPv4-compatível. Mas existem ainda os túneis *6to4*, ISATAP, e TEREDO, cujas formas de endereçamento são diferentes umas das outras [8].

Aqui, iremos nos deter apenas à explanação de túneis *6to4* [8][18]. Para aprofundar-se no tunelamento automático com a utilização de endereços IPv6 do tipo IPv4-compatível, podem ser consultadas a RFC 2893, ou a RCF 4213 – proposta recentemente para substituir a anterior. Túneis ISATAP e TEREDO estão ainda em estágio de experimentação, e são descritos respectivamente através das RFCs 4214 e 3904.

A técnica de tunelamento *6to4*, assim como outros tipos de tunelamento, é uma medida provisória, que só deverá existir enquanto for necessária a sua utilização para facilitar a migração para o IPv6.

Como descrito na RFC, o método de acesso *6to4* é um método de acesso para a 'conexão de domínios IPv6 via nuvens IPv4'. A infra-estrutura IPv4 é encarada como uma conexão *unicast* ponto-a-ponto, e os domínios IPv6 comunicam-se através de roteadores *6to4*, chamados de *6to4 gateways*, que são os responsáveis pelo encapsulamento e desencapsulamento dos pacotes [57].

O IANA destinou, para o endereçamento *6to4*, um TLA específico. A Figura 8 mostra em detalhes o formato de endereços *6to4*. O prefixo reservado é *2002::/16*. Os 32 *bits* seguintes são preenchidos com o endereço IPv4 do *gateway* da rede. Esse IPv4 deve ser um IP real, isto é, um IP roteável na Internet. Os 80 *bits* restantes são utilizados para definição de endereço da rede local, e dos *hosts* da rede. A divisão desses bits não é pré-fixada. Pode-se tomar, por exemplo, 16 *bits* para a rede e 64 *bits* para as interfaces dos *hosts* da rede, como mostrado na figura. Os primeiros 48 *bits* são então considerados como prefixo do endereço, e uma vez configurados não mudam – a não ser que o IPv4 do *gateway* sofra alteração.

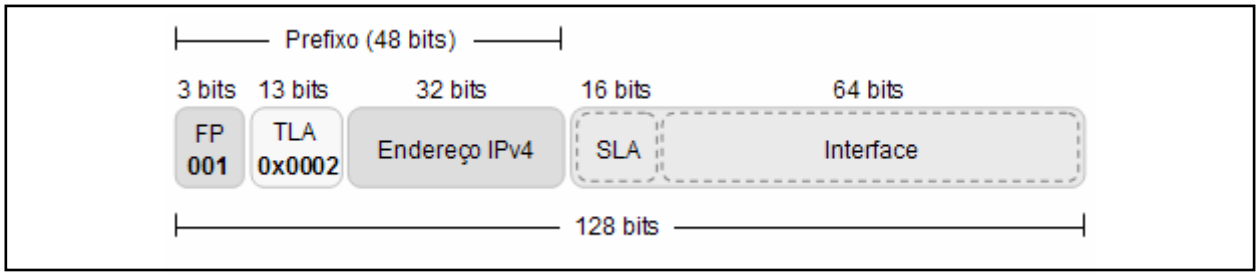

Figura 8. Formato de um endereço IPv6 do tipo *6to4*.
Fonte: HAGEM, SILVIA. **IPv6 Essentials**. California: O'Reilly & Associates, 2002. p. 38.

O *6to4* tem se tornado um meio de acesso IPv6 muito adotado por organizações que desejam conectar-se à rede 6Bone – o ramo principal da rede IPv6 atual, mas que não possuem ainda um IPv6 válido por não estarem atreladas a um provedor de serviços (ISP) que ofereça suporte a IPv6. Existem roteadores específicos para permitir o acesso indiscriminado à rede IPv6 do 6Bone, que são chamados de *6to4 relay routers*. Esses equipamentos estão diretamente conectados ao *backbone* IPv6, e possuem também endereços IPv4.

Existem poucos *relay routers*, mas a tendência é que com a popularização do acesso IPv6, a quantidade aumente de forma equivalente. Cada um desses roteadores possui um IPv4 válido, que pode ser referenciado manualmente. No entanto, a RFC 3068 define uma faixa de endereços, *192.88.99.0/24*, chamado *prefixo de endereço anycast 6to4 relay router*, na qual o primeiro endereço válido, ou seja, *192.88.99.1*, é o endereço *anycast 6to4 relay router* propriamente dito. É a existência desse endereço *anycast* especial que classifica os túneis *6to4* como um tipo de

tunelamento automático. Existem motivos para a reserva de toda uma faixa de IPs, descritos nessa mesma RFC.

Esse endereço especial facilita a configuração dos túneis *6to4*. Se não existisse, uma organização teria que conhecer previamente o endereço de um *6to4 relay router*, e se esse roteador falhasse, o acesso à rede 6Bone teria que ser reconfigurado. A função do endereço reservado pelo IANA é servir como uma espécie de endereço global, que substitui os endereços dos *relay routers* na configuração dos pontos de entrada dos túneis. Ao ser enviado para esse endereço, um pacote é redirecionado para o *relay router* mais próximo. Se esse roteador, por algum motivo, deixar de funcionar, os pacotes são automaticamente redirecionados para um outro *relay router*, novamente o mais próximo à origem. A principal desvantagem da utilização desse endereço é o custo de tempo adicional, introduzido pelo processo de escolha do roteador mais próximo, que decrementa o desempenho da conexão.

4.3.2 Tunelamento configurado

O tunelamento com configuração explícita exige a pré-configuração do ponto de saída do túnel, tornando-o um pouco mais complexo do que o tunelamento automático. Em compensação, possui algumas vantagens.

No tunelamento automático, não há como ter certeza da idoneidade do roteador de destino, pois não há processo algum de autenticação e reconhecimento das extremidades do túnel. Já o tunelamento configurado permite, entre outras opções, a existência de tal processo, o que garante uma maior segurança tanto para o usuário – que tem a certeza de que seus dados estão sendo manipulados por um roteador com fins benéficos, quanto para o servidor do acesso – que através da autenticação elege os usuários autorizados a enviar mensagens entre as redes IPv4 e IPv6, evitando assim ataques como os de *negação de serviço* [9], por exemplo.

A RFC que define os conceitos de tunelamento configurado é a de número 2893, e é a mesma que define o tunelamento automático. Mas na RFC apenas se estabelecem e se apresentam os conceitos. Não há determinação de nenhum mecanismo que implemente tais definições. Fica a cargo de outras RFCs, definirem protocolos e ferramentas que ponham em prática as idéias expostas.

O *freenet6* [52] é um serviço de acesso IPv6 grátis, provido por uma empresa canadense pioneira na fabricação de equipamentos de migração IPv6, a HEXAGO. Essa empresa, que está conectada com a rede experimental do 6BONE, disponibiliza esse serviço de tunelamento explicitamente configurado, cuja arquitetura está descrita na RFC 3053. Foi criado um protocolo, o TSP (*Tunnel Setup Protocol*), para a negociação de um túnel com o roteador de acesso ao IPv6, o qual é chamado de *Tunnel Broker*. Uma ferramenta chamada Cliente TSP deve ser instalada no computador do usuário, pois é a responsável por repassar as informações de configuração do cliente para o *tunnel broker*, a fim de adquirir um endereço e um túnel IPv6.

A autenticação do usuário, com o *freenet6*, não é obrigatória. Se um cliente é um *host* único, e se conecta sem autenticação, como anônimo, recebe um endereço IPv6 global do tipo *2001:5c0:8fff:ffff::/64*. Se a autenticação é adotada, o endereço atribuído é da forma *2001:5c0:8fff:fffe::/64*. Se o cliente for uma organização, que deseja disponibilizar acesso a parte de sua rede interna, ele pode requerer a delegação de um prefixo de 48 *bits*, que deve ser alocado para o roteador/*gateway* IPv6 da organização, que por sua vez se encarregará de distribuir os endereços pela rede local.

O processo de autenticação pode ser realizado através de texto plano ou de criptografia MD5. O *freenet6* provê, ainda, a delegação de registros DNS e DNS reverso, além de outras facilidades.

Capítulo 5

Aspectos de segurança em IPv6

Questões que dizem respeito à segurança do protocolo IPv6, e dos demais protocolos dependentes dessa nova versão do protocolo IP, têm se mostrado um pouco controversas, e gerado bastante polêmica. Muitos acreditam que a adoção da estrutura IP Seguro (IPSEC) [18][34][37] deverá ser o fim dos problemas de segurança existentes hoje com a versão 4 do protocolo IP.

Na verdade, o IPSEC – que apesar de desenvolvido originalmente para IPv6, tem sido portado e adotado para IPv4, com algumas ressalvas – só provê medidas de segurança na camada de rede, o que de qualquer forma já é um ponto de partida. O IPSEC na verdade é uma estrutura formada por vários componentes com finalidades específicas, e que se integram para chegar a um resultado comum: garantir a integridade, a confidencialidade e a autenticidade dos pacotes IP.

A estrutura do IPSEC é composta por seis elementos distintos [18]:

- Uma descrição geral de requisitos de segurança e mecanismos, na camada de rede;
- Um elemento de segurança específico para cifragem, o ESP (*Encrypted Security Payload*);
- Um elemento de segurança específico para autenticação, o AH (*Authentication Header*);
- Definições para uso concreto de algoritmos criptográficos para cifragem e autenticação dos dados;
- Definições de políticas e associações de segurança entre sistemas comunicantes;
- Adoção de uma estrutura para administração e distribuição de chaves para o IPSEC.

A forma como esses componentes interagem entre si é mostrada na figura a seguir. A vantagem dessa estrutura é que as mensagens podem ser submetidas separadamente aos componentes ESP e AH, ou a ambos, caso seja necessário. O protocolo de autenticação utilizado e definido como padrão é o *Keyed* MD5. O protocolo de cifragem de dados é o DES, mas o 3DES pode também ser adotado [36].

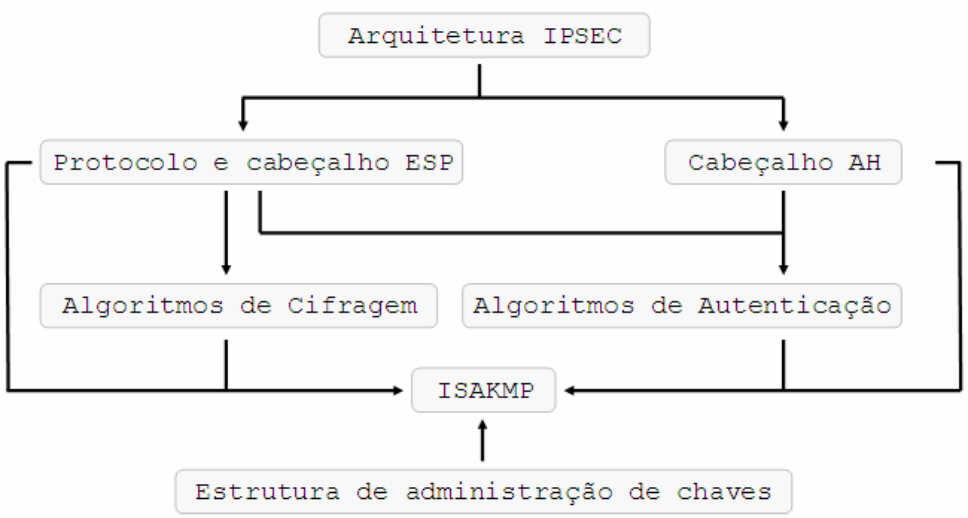

Figura 9. Relacionamento entre os componentes do IPSEC.
Fonte: HAGEM, SILVIA. **IPv6 Essentials**. California: O'Reilly & Associates, 2002. p. 89.

Apesar de prover um certo nível de segurança, o IPSEC não exclui a necessidade de algumas ferramentas, como *firewall* e IDS (*Intrusion Detection System*) [13][34][35], por exemplo. No entanto, com a utilização desse conjunto de protocolos será possível eliminar falhas de segurança intrínsecas a protocolos como FTP, DNS, Telnet, dentre outros. Assim, poderiam cair em desuso alguns outros protocolos, como por exemplo o *Secure Shell* (SSH), que surgiu para prover maior segurança para conexões remotas, em substituição ao Telnet.

Uma das maiores confusões que se faz diz respeito ao destino do protocolo SSL (*Security Socket Layer*). Geralmente se costuma entender que com a adoção definitiva do IPSEC, o SSL deixará de ser necessário, mas essa não é uma afirmação totalmente correta [17]. Essa confusão acontece porque, do ponto de vista do usuário, os serviços oferecidos pelo SSL são parecidos com aqueles a que o IPSEC se dispõe a resolver: autenticação e cifragem de dados.

O IPSEC ainda vem sendo motivo de muito estudo, e não está nem perto de chegar a uma implementação final. Várias questões de segurança e de compatibilidade estão ainda em aberto, e precisam evoluir [16][19]. Uma infra-estrutura de chaves públicas (*Public Key Infrastructure*) eficiente ainda precisa ser definida e adotada.

O SSL, um protocolo de camada mais superior, que se apóia no TCP, e que foi designado primeiramente para tratar questões de segurança relativas ao protocolo HTTP, deverá, portanto, continuar atuando ainda por um bom tempo. Afinal, permite maior transparência no processo de troca de chaves e certificados, em relação ao IPSEC, que ainda se encontra em estágio de amadurecimento.

No entanto, em longo prazo, assumindo-se uma evolução nas definições do IPSEC, o SSL deverá ser cada vez menos utilizado, restringindo-se a casos específicos como transações com números de cartões de crédito, por exemplo. Afinal, por estar localizado em uma camada mais inferior, o IPSEC acaba se tornando uma solução generalizada, que pode ser utilizada por todos os protocolos das camadas superiores, sem distinção.

Um dos maiores empecilhos no desenvolvimento do IPSEC é que ele vem se tornando cada vez mais complexo, dificultando o entendimento e resolução dos possíveis problemas que possam vir a ocorrer devido à sua utilização [18].

O uso do IPSEC, e também de ferramentas como um *firewall*, é importante no IPv6, principalmente porque não há mais distinção entre endereços privados e reais, não há mais NAT. Agora, todo computador conectado à Internet possui um endereço acessível de qualquer lugar do mundo, e se não for devidamente protegido, com certeza será invadido.

Capítulo 6

6ix Linux: um facilitador do acesso à Internet IPv6

Uma Distribuição Linux nada mais do que é um sistema operacional, composto por um conjunto de programas básicos e utilitários, e que utiliza como núcleo do sistema o *Kernel* Linux, desenvolvido originalmente por Linus Torvalds, e atualmente mantido pelo próprio e por um conjunto de profissionais [12][40][66]. O *Kernel* Linux e as distribuições baseadas nele são *softwares* livres, em sua maioria baseados na licença GPL (*General Public License*) do projeto GNU [12][40][76]. A diferença entre as distribuições está no conjunto de ferramentas que as compõem, em alguns utilitários exclusivos de cada uma, e em particularidades como arquivos de configurações e suas localizações.

Há até bem pouco tempo, configurar uma distribuição linux para trabalhar com o protocolo IPv6 era uma tarefa árdua. As versões mais atuais do *Kernel* não suportavam o protocolo. Era preciso então utilizar algum conjunto de pacotes desenvolvido isoladamente por grupos de estudo, como por exemplo o do projeto Kame [64], e passar por todo um processo complexo de compilação e adaptação do sistema, para que finalmente uma rede local pudesse se comunicar através do novo protocolo.

Isso acontecia porque a disseminação do IPv6 ainda era pouca, e a sua importância praticamente nula para a maioria dos usuários e até para os desenvolvedores do *Kernel*. Essa realidade começou a mudar quando o debate sobre a iminente exaustão de endereços IPv4 se intensificou, acelerando a intenção de migrar para o novo protocolo. Grupos como o do projeto Kame tiveram participação importante nessa corrida, por já estarem em estágio avançado de desenvolvimento dos padrões, e por ajudarem a divulgar as características do IPv6, facilitando o trabalho dos desenvolvedores do *Kernel* Linux. Felizmente, hoje praticamente todas as distribuições linux já possuem suporte ao novo protocolo IP, que já vem por padrão integrado às versões mais recentes do núcleo das distribuições.

Uma das distribuições linux mais famosas é o Debian GNU/Linux [21][29] [48], que vem com mais de 15.490 pacotes contendo *softwares* pré-compilados e distribuídos em um bom formato, facilitando a instalação do sistema. O Debian também é conhecido como a distribuição linux mais estável, já que antes de um pacote ser atualizado uma bateria exaustiva de testes é efetuada. Outra característica importante do Debian é a presença de uma ferramenta chamada *apt-get*, que facilita bastante a instalação de novos pacotes, praticamente automatizando essa tarefa, tornando o sistema comparável a outros sistemas operacionais proprietários, conhecidos pela

facilidade e transparência na instalação de programas [29]. A versão mais atual do Debian é a 3.1, apelidada de Sarge.

As grandes distribuições linux são poucas. As demais distribuições geralmente tomam como base de desenvolvimento uma dessas grandes distribuições, absorvendo assim a maioria de suas características, e algumas vezes acrescentando novidades. Uma distribuição baseada no Debian é conhecida como *Debian-based*.

O 6ix Linux é uma distribuição linux *Debian-based*, que tem como finalidade divulgar e intensificar a utilização do protocolo IPv6, e algumas das mudanças por ele introduzidas. Afinal, reconhecer o protocolo IPv6 por si só não é suficiente. Se o computador ainda não está localizado em uma rede com acesso direto a IPv6 – que é a realidade da maioria das organizações e dos usuários domésticos, de nada adianta a capacidade de trabalhar com o novo protocolo IP. É preciso então adotar algum método de migração que permita o acesso à Internet IPv6.

O 6ix Linux disponibiliza duas maneiras de obter acesso à rede IPv6, ambas através de túneis. É possível escolher entre a configuração de um túnel automático *6to4*, ou a de um túnel *freenet6* explicitamente configurado. Além dessas duas ferramentas de migração, o 6ix Linux conta ainda com uma gama de *softwares* preparados para possibilitar a implementação de uma rede local onde os computadores se comuniquem através do IPv6, como serviços de roteamento, uma calculadora, ferramentas para depuração e análise do comportamento da rede, uma aplicação para facilitar a configuração de um *firewall*, dentre outros.

O ambiente gráfico escolhido para a distribuição foi o KDE [65], um gerenciador de aplicações gráfico poderoso, estável e desenvolvido totalmente em C++. O KDE, juntamente com o GNOME, é o gerenciador gráfico mais conhecido para ambientes linux, e tem evoluído bastante nos últimos anos, sendo adotado como padrão por grande parte das distribuições linux existentes. A Figura 10 mostra a tela do ambiente de trabalho da versão final do 6ix Linux.

Figura 10. Ambiente de trabalho do 6ix Linux.

O 6ix Linux foi desenvolvido a partir do Kalango Linux [63], que por sua vez surgiu como uma customização do Kurumin Linux [69]. Ambos são projetos brasileiros, baseados no Knoppix Linux [67], um projeto *Debian-based* pioneiro na capacidade de executar a partir do CD-ROM, sem necessitar propriamente de instalação no disco rígido do computador. Distribuições assim ficaram conhecidas como *Live-CD*. Sendo assim, por hereditariedade, o 6ix Linux constitui-se em uma Distribuição Linux *Live-CD Debian-based*, assumindo as características de todos os seus ancestrais.

A vantagem de sistemas *Live-CD* é permitir a adoção de um sistema operacional exigindo o mínimo de configuração adicional no computador e evitando perda tempo com instalação. O *hardware* da máquina é reconhecido e configurado automaticamente, durante o processo de inicialização. Assim, pode-se conhecer de forma mais aprofundada o sistema, bem como suas características e facilidades, sem alterar nem causar danos aos demais sistemas operacionais instalados na máquina.

6.1 Fases de desenvolvimento

O 6ix Linux foi desenvolvido a partir de uma distribuição já existente, poupando tempo de desenvolvimento e configuração de componentes básicos do sistema, e possibilitando uma concentração maior de esforço na produção da capacidade de acesso à rede IPv6 e na instalação das ferramentas de configuração de rede local. É possível identificar várias etapas distintas na construção da distribuição.

Por tomar como base uma distribuição pré-existente, fica claro que o primeiro passo no desenvolvimento do 6ix Linux foi a customização. Foi preciso modificar o ambiente, tanto visualmente quanto em diversos arquivos de configuração, dando à distribuição uma identidade própria.

Em seguida, veio a fase de remoção e adição de pacotes. Pacotes previamente instalados – mas julgados desnecessários, tiveram de ser removidos, para evitar desperdício de espaço de armazenamento. Afinal, o 6ix Linux deveria caber em um CD-ROM. Ao mesmo tempo, foram instalados novos pacotes, necessários ao bom funcionamento do sistema, ou que pelo menos acrescentassem alguma facilidade.

Essas duas etapas iniciais constituem a fase de preparação do sistema. Acabada essa fase de preparação, foi preciso obter o acesso à rede IPv6. A versão do *Kernel* utilizado no 6ix linux foi a 2.6, que já suporta implicitamente a nova pilha de protocolos do IPv6 [66]. Mas, como dito anteriormente, não basta o sistema operacional lidar com o protocolo, era preciso ainda encontrar um modo de se comunicar através do mesmo. O acesso à rede do 6Bone, no 6ix Linux, é garantido através de uma ferramenta gráfica, desenvolvida para facilitar a configuração do acesso IPv6.

As duas últimas etapas no desenvolvimento da distribuição foram a criação de uma ferramenta de configuração do firewall e a instalação de pacotes específicos para a configuração de ambientes de rede local com suporte a IPv6, como ferramentas de roteamento e auto-configuração, por exemplo, além de outros componentes com suporte a IPv6, como é o caso do servidor *web* instalado.

Terminada a criação da distribuição, partiu-se para a etapa de testes e avaliação de comportamento da mesma. Duas máquinas pertencentes a redes distintas foram utilizadas para, através do 6ix Linux, estabelecerem uma comunicação IPv6.

Por fim, foram realizados experimentos para analisar o desempenho dos dois métodos de acesso IPv6, visando à comparação entre os métodos, e uma apreciação final das vantagens e desvantagens de cada um.

As próximas seções descrevem em detalhes, cada etapa mencionada acima, apresentando inclusive os percalços que tiveram que ser enfrentados e vencidos, introduzindo conceitos relacionados, e demonstrando o processo de desenvolvimento e análise da distribuição propriamente ditos.

6.2 Customização do ambiente

A etapa de customização da distribuição é importante, por ser o ponto de partida do projeto. Além disso, criar uma identidade própria para a distribuição é importante para divulgá-la. Afinal, a apresentação visual bem acabada de um software é uma ferramenta de marketing indispensável, pois ajudar a conquistar clientela, atraindo com mais eficiência a atenção do público-alvo do sistema.

Essa etapa começa com a preparação do ambiente de desenvolvimento da distribuição, que serve também para as demais etapas. Analisando a árvore de diretórios do CD-ROM da distribuição Kalango Linux, encontramos duas pastas: *boot* e *knoppix*. A pasta *boot* é onde ficam os arquivos responsáveis pela inicialização do sistema diretamente a partir do CD. Na pasta *knoppix*, existe um arquivo que ocupa a maior parte do CD. Este arquivo nada mais é do que uma imagem compactada da partição raiz do sistema. Todos os arquivos da distribuição são compactados neste arquivo, permitindo melhor aproveitamento do espaço de armazenamento. Assim, em um CD de 700 MB é possível gravar um sistema com cerca de 2 GB de informações.

Durante a inicialização, entra em ação uma ferramenta chamada de *Módulo Cloop*, que descompacta alguns módulos essenciais do sistema, e engana o *Kernel*, fazendo-o pensar que está acessando uma partição do disco rígido. Quando é preciso carregar um arquivo dentro de uma pasta, por exemplo, esse módulo lê e descompacta o arquivo repassando esses dados para o *Kernel*. Esse *Módulo Cloop* foi originalmente desenvolvido por Andrew Morton, atual mantenedor do *Kernel* 2.6 do Linux. Na época, ele não conseguiu achar funcionalidade para a ferramenta, deixando-a de lado. Algum tempo depois, Klaus Knopper, soube da existência do módulo e o utilizou como um dos componentes básicos de sua distribuição, o Knoppix, dando origem à primeira distribuição linux *Live-CD*.

Quando uma distribuição *live-CD* está em execução, algumas pastas do sistema que precisam de suporte a escrita, como os diretórios dos usuários, por exemplo, são armazenadas num espaço na memória RAM, com cerca de 2 MB, que é criado durante a inicialização. Este espaço de memória reservado pode crescer conforme necessário, desde que exista memória suficiente. O sistema usa ainda, para poupar espaço na memória RAM, partições *linux swap*, ou arquivos de troca encontrados em partições Windows, caso exista alguma no disco rígido da máquina.

Para gerar uma versão personalizada do sistema, é preciso descompactar o arquivo da imagem numa pasta do disco rígido, fazer as modificações desejadas, gerar uma nova imagem compactada e finalmente preparar um novo CD. Esse processo é conhecido como remasterização. Alguns dos pré-requisitos para remasterizar distribuições como o Knoppix e o Kalango, são uma partição linux devidamente formatada, com cerca de 4 GB não utilizados, e uma memória *swap*, ou equivalente, que somada à memória RAM do computador alcance 1 GB de capacidade, pelo menos.

O Kalango Linux disponibiliza uma ferramenta para automatizar algumas etapas importantes do processo de remasterização, como mostra a Figura 11. No entanto, todo o processo pode ser realizado manualmente, com privilégios de administrador do sistema [32][69].

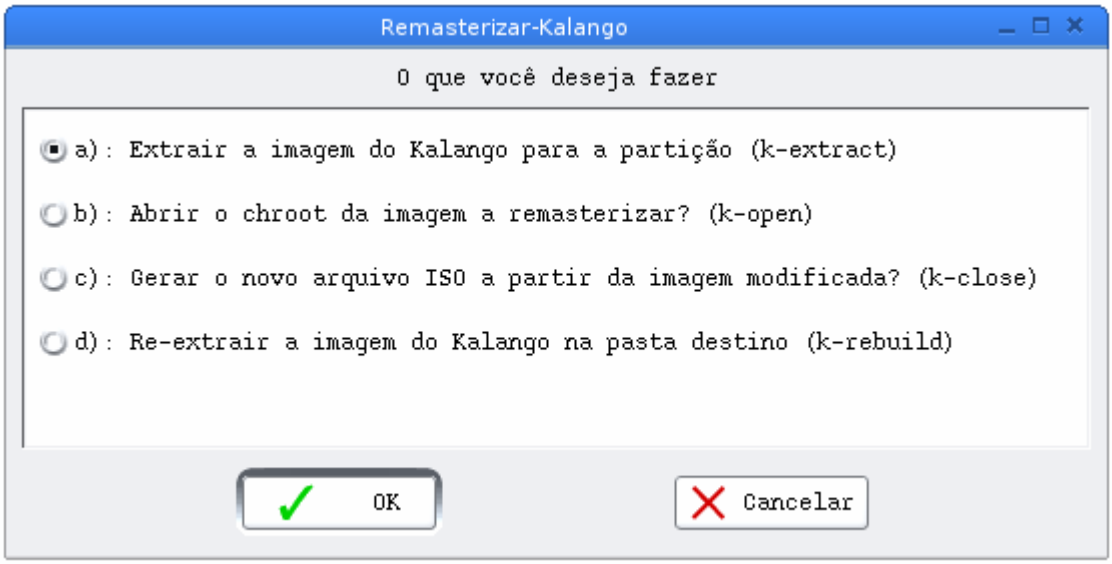

Figura 11. Script de remasterização do Kalango Linux.

Para realizar a etapa de extração dos arquivos manualmente, primeiro logamos em um terminal como superusuário do sistema, o root, através do comando ***su -***. A senha padrão para o superusuário do 6ix Linux é *pedrofrancaneto*. Em seguida, montamos a partição onde os arquivos seriam armazenados, neste caso a partição *hda4*, através do comando ***mount -t reiserfs /dev/hda4 /mnt/hda4***. O formato de partição escolhido foi o *ReiserFS*, que é o padrão da distribuição.

A seguir, foi preciso criar dentro da partição montada os diretórios onde seriam armazenados os arquivos, utilizando o comando ***mkdir***. O diretório chamado *knxmaster* contém arquivos relacionados ao processo de *boot* do sistema. Já no diretório *knxsource*, encontra-se a pasta *KNOPPIX*, com todos os arquivos do sistema propriamente dito. A maioria das modificações é realizada dentro da pasta *knxsource*. No entanto, alterações na pasta *knxmaster* também se fazem necessárias.

A última etapa do processo de extração dos arquivos foi finalmente copiar e extrair o arquivo de imagem do sistema para a pasta */knxsource/KNOPPIX*, através do comando ***cp -Rp /KNOPPIX/* /mnt/hda4/knxsource/KNOPPIX***. A partir daí, passou-se à etapa de edição de arquivos.

As modificações realizadas no sistema original, para a criação do 6ix Linux, tiveram os mais variados fins. Algumas modificações tiveram cunho visual, alterando telas e mensagens mostradas durante a inicialização e na área de trabalho da distribuição. Outros arquivos foram modificados para permitir que o novo sistema pudesse funcionar adequadamente. Basicamente, a quase totalidade dos esforços de alteração foi concentrada em duas pastas localizadas na raiz do sistema: */etc* e */usr*. A pasta */etc*, em sistemas linux, tem como função armazenar arquivos de configuração local, tais como informações de usuários, grupos e senhas do sistema, e serviços utilizados durante o processo de inicialização do sistema. A pasta */usr*, por sua vez, contém arquivos de programas utilizados frequentemente pelos usuários do sistema, como, por exemplo, a ferramenta de configuração de acesso ao IPv6, ou um editor de textos, ou mesmo um editor de imagens.

A quantidade de arquivos criados, modificados e apagados foi bastante significativa – muito mais de 100 arquivos, podendo ser classificados em vários níveis de complexidade. Em alguns, bastou apenas efetuar modificações pontuais, como nomes de pastas ou de arquivos referenciados, por exemplo. Outros exigiram processos mais complexos, como alterações de formatos ou de permissões de acesso às pastas. Para mudar a imagem que aparece durante o boot

do sistema, por exemplo, foi preciso gerar uma imagem em formato PNG (*Portable Network Graphics*), converter a imagem para um formato intermediário, e finalmente converte-la para o formato final, mais compacto e com menor número de cores, utilizado pelo sistema. Tudo isso, com a ajuda de ferramentas de conversão, que tiveram que ser instaladas temporariamente.

Por questões de segurança, o processo de remasterização do 6ix Linux foi repetido constantemente, sempre que alguns arquivos eram modificados. Assim, se alguma alteração acarretasse na falha geral do sistema, um ponto de recuperação, com pouca diferença, poderia ser utilizado. Essas imagens de *backup* eram devidamente identificadas, e armazenadas em uma outra partição linux do disco rígido da máquina. Também era gravado um CD com a imagem mais recente do sistema.

Na verdade, o processo de remasterização é contínuo, e só se encerra quando todas as etapas de desenvolvimento da nova distribuição são efetuadas, gerando finalmente um produto final e encerrando o ciclo de modificações e remasterizações.

6.3 Manipulação de pacotes básicos

Uma outra etapa inicial, preparatória para a instalação e criação das ferramentas IPv6, foi a manipulação de pacotes básicos do sistema. Pacotes considerados desnecessários foram removidos, para poupar espaço de armazenamento e diminuir o tamanho da imagem do sistema. Paralelamente, alguns pacotes fundamentais, necessários para a instalação dos demais pacotes e ferramentas, foram instalados.

Para desinstalar os pacotes de jogos e outros programas desnecessários, foi utilizada a ferramenta Synaptic, um gerenciador gráfico de pacotes presente no Kalango. A vantagem desse gerenciador é a organização de pacotes, tornando mais intuitivo o trabalho de remoção.

Alguns pacotes foram removidos manualmente, através do comando ***apt-get remove***. Antes de prosseguir com a remoção, é feita uma checagem de dependências e uma pergunta sobre a intenção de ir adiante, como mostrado no quadro a seguir.

Quadro 1. Execução do comando *apt-get remove* em um terminal no Kurumin Linux.

```
root@kurumin:/home/kurumin# apt-get remove kdelibs4
Lendo Lista de Pacotes... Pronto
Construindo Árvore de Dependências... Pronto
Os pacotes a seguir serão REMOVIDOS:
  ark k3b k3blibs kaffeine kaffeine-mozilla kappfinder karamba kate katomic
  kaudiocreator kcalc kcharselect kcontrol kde-i18n-ptbr kdeartwork-style
  kdeartwork-theme-window kdebase-bin kdebase-kio-plugins kdelibs kdelibs-bin
  kdelibs4 kdemultimedia-kio-plugins kdenetwork-kfile-plugins kdeprint
  kdesktop kdf kdict kdm kedit kfind kfloppy kgamma kget kghostview kicker
  kicker-applets klaptopdaemon kmail kmailcvt kmenuedit knotes koffice-data
  koffice-libs kommander konq-plugins konqueror konqueror-nsplugins konsole
  kopete kpackage kpf kpilot kppp krdc krfb kscd ksmserver ksnapshot ksokoban
  kspread ksysguard ktnef kuickshow kwin kword kxconfig libkcal2 libkcddb1
  libkdeedu1 libkdegames1 libkdenetwork2 libkdepim1 libkonq4 libksieve0
  plastik sim
0 pacotes atualizados, 0 novos instalados, 76 a serem removidos e 357 não atuali
zados.
É precis fazer o download de 0B de arquivos.
Depois de desempacotar, 172MB de espaço em disco serão liberados.
Quer continuar? [S/n]
```

Após a remoção de pacotes, é comum que restem alguns poucos pacotes isolados, sem mais nenhuma serventia chamados de pacotes órfãos. Para eliminar esses pacotes, pode se utilizar o comando ***deborphan***, que retorna uma lista de pacotes órfãos, se houver algum, que podem ser removidos com segurança, sem afetar o sistema.

Quando se utiliza o *apt-get*, os pacotes são armazenados em um espaço do disco, uma espécie de memória cache de pacotes. Para remover esses arquivos, utiliza-se o comando ***apt-get clean***.

A instalação dos novos pacotes é mais simples. Essa tarefa foi realizada através da ferramenta *apt-get*. Pacotes instalados através do *apt-get* são obtidos de repositórios públicos, por HTTP ou FTP, previamente indicados em um arquivo. Ao serem armazenados na máquina, seguem uma série de comandos, e o direcionamento de arquivos e alterações do sistema são feitos praticamente de forma automatizada, deixando as ferramentas prontas para utilização [29]. Os seguintes pacotes foram instalados:

- **iputils-arping:** ferramentas para enviar mensagens ICMP do tipo *echo request* para endereços ARP. Suporta IPv6;
- **iputils-ping:** ferramentas para testar a disponibilidade das máquinas de uma rede através de mensagens ICMP dos tipos *echo request* e *echo reply*. Suporta IPv6;
- **iputils-tracepath:** ferramentas para traçar o caminho pela rede até um dado destino, inclusive descobrindo os MTUs de cada enlace. Suporta IPv6;
- **ipv6calc:** pequeno utilitário para manipulação de endereços IPv6, calculando-os em diversos formatos.
- **tspc:** cliente utilizado para configurar um túnel IPv6 *freenet6*.

Após essas duas fases do desenvolvimento da distribuição 6ix Linux, foi realizado um teste de comportamento do sistema. O teste consistiu na utilização do sistema por um período de dois dias, tentando encontrar algum mal-funcionamento. Nenhuma anomalia aparente foi encontrada. Posteriormente, foi gerado um CD, ao qual se deu o nome de *primeira versão estável*. Essa versão foi utilizada para dar seguimento ao desenvolvimento da distribuição.

6.4 Criando a ferramenta de acesso IPv6

O 6ix Linux, como já foi citado anteriormente, disponibiliza dois métodos distintos de acesso ao *backbone* IPv6, ambos através da técnica de tunelamento. Para facilitar a configuração do acesso IPv6, foi desenvolvida uma ferramenta gráfica, por ser mais amigável do que uma ferramenta de linha de comando em modo texto.

Na verdade, essa ferramenta é composta por duas partes: o *front-end*, uma interface gráfica desenvolvida em Kommander [68]; e o *back-end*, composto por *scripts* desenvolvidos em Python [77], responsáveis pela manipulação de arquivos e configuração do sistema. Tais *scripts* também podem ser utilizados diretamente via linha de comando, por usuários mais experientes.

A parte principal da ferramenta, como pode ser facilmente percebido, é o *back-end*, composto pelos *scripts* de configuração. Foram desenvolvidos dois *scripts* distintos, um para configurar o acesso através de um túnel *6to4*, e outro para configurar o acesso através de um túnel *freenet6*. Além desses *scripts*, desenvolvidos em Python, foi criado também um pequeno *script* em linguagem *shell* [12], chamado *ipv6.sh*, com a função de calcular o endereço IPv6 a partir de um determinado endereço IPv4. Este *script* simples, cujo código é mostrado no Quadro 2, é utilizado como auxiliar na configuração do túnel *6to4*.

Quadro 2. Código do *shell script ipv6.sh*.

```
#!/bin/sh
printf "2002:%x%02x:%x%02x::\n" `echo $1 | sed 's/\./ /g'`
```

Esses três arquivos de *scripts* foram colocados no diretório */usr/local/bin* do sistema. Como essa pasta já faz parte, por padrão, do *path* de execução do sistema, a execução de tais *scripts* é feita de forma muito mais simples, pois os mesmos passam a ser reconhecidos em todo o sistema. Para dar um exemplo, vamos pegar o caso do *script shell*, chamado *ipv6.sh*. Se não fizesse parte do *path*, para ser executado seria necessário navegar até a pasta em que está localizado, e a seguir digitar o comando ***./ipv6.sh***. No entanto, por fazer parte do caminho do sistema, basta agora digitar ***ipv6.sh*** na linha de comando, dentro de qualquer diretório, que o *script* será executado.

O que o *script* de configuração do túnel *6to4* faz é calcular o endereço IPv6 de uma interface a partir do seu endereço IPv4. Em seguida, manipula o arquivo */etc/network/interfaces*, adicionando algumas linhas específicas ao arquivo, para permitir a configuração do túnel IPv6. Existem ainda outras funções secundárias desempenhadas pelo *script*.

As linhas adicionadas ao arquivo citado são as mostradas no Quadro 3. A primeira linha serve para especificar a criação da interface (*iface*), chamada *sit1*, do tipo IPv6 (*inet6*) e que é tunelada no IPv4 (*v4tunnel*). A segunda e terceira linhas especificam o endereço IPv6 da interface e o prefixo. A quarta linha define o endereço IPv4 do ponto de saída do túnel (*endpoint*). Como esse endereço será definido pelo *anycast 6to4 relay router*, não é preciso especificar o final do túnel. A linha começada com *local* especifica o endereço IPv4 da interface. A função das linhas começadas com *up* e *down* é passar para o sistema tarefas que devem ser efetuadas ao se levantar (*up*) uma interface, isto é, ligar a interface, ou ao se desligar a interface (*down*). Neste caso, as linhas adicionam e removem uma entrada da tabela de roteamento da máquina, que aponta justamente para o endereço reservado *anycast 6to4 relay router* (*192.88.99.1*), que serve como *gateway* do túnel. A última linha indica apenas o maior número de saltos IPv6 possíveis a serem percorridos pelos pacotes. A rede IPv4 usada pelo túnel é considerada apenas como um simples salto, ou enlace, portanto seriam necessários 64 saltos IPv6 para que um pacote venha a ser descartado, prevenindo a ocorrência de *loops* de roteamento.

Quadro 3. Linhas adicionadas ao arquivo */etc/network/interfaces*, para configurar o túnel *6to4*.

```
iface sit1 inet6 v4tunnel
 address 2002:a14:1e28::2
 netmask 128
 endpoint any
 local 10.20.30.40
 up ip -6 route add 2000::/3 via ::192.88.99.1 dev sit1
 down ip -6 route flush dev sit1
 ttl 64
```

A próxima figura mostra a saída da execução do comando ***6to4***, sem parâmetros, em um terminal de modo texto, que exibe a forma de utilização correta do *script*.

```
6to4: parâmetros incorretos.
Modo de uso: 6to4 <interface> <auto> <firewall> | <command>
  onde:
   <interface> Interface de conexão que possui um IP válido. Ex: ppp0
   <auto>      Define se a configuração passa a ser automática a  partir do
               próximo boot. Possíveis valores: True , False
   <firewall>  Define se o firewall será configurado com regras específicas
               para IPv6 ou não. Possíveis valores: True , False
   <command>   Executa um comando específico. Possíveis valores:
                 clear - Limpa as configurações do túnel 6to4
                 start - 'Liga' a interface sit1 ('abre' o túnel)
                 stop  - 'Desliga' a interface sit1 ('fecha' o túnel)
```

Figura 12. Saída exibida na tela, ao ser executado o comando ***6to4*** em modo texto.

Já o *script* de configuração do túnel *freenet6*, é mais simples. Ele serve de interface para o usuário perante a ferramenta *TSP Client* disponibilizada pelo serviço da HEXAGO. O *script* desenvolvido formata a entrada de dados repassados para o cliente TSP, abstraindo para o usuário algumas configurações necessárias. O cliente TSP pode inclusive ser utilizado, com mais opções, por usuários mais experientes, mas somente em modo texto.

A função do cliente TSP é parecida com o processo descrito a pouco, para o *6to4*. Ele modifica alguns arquivos, objetivando configurar uma interface de túnel, e adicionando rotas à tabela da máquina. No entanto, muitas dessas configurações, como por exemplo o endereço IPv6 a ser utilizado, são obtidas através de um processo de negociação com o *Tunnel Broker*. O quadro abaixo mostra o XML (*Extensible Markup Language*) retornado pelo servidor durante uma transação de túnel, com configurações padrões, que é utilizado pelo cliente para retirar as informações necessárias à montagem local do túnel, como o próprio endereço IPv6 delegado e o IPv6 do servidor.

Quadro 4. XML contendo dados da configuração do túnel *freenet6*.

```
sent: Content-length: 203
<tunnel action="create" type="v6anyv4" proxy="no">
 <client>
  <address type="ipv4">201.8.254.18</address>
<keepalive interval="30"><address type="ipv6">::</address></keepalive> </client>
</tunnel>

recv:
200 Success
<tunnel action="info" type="v6v4" lifetime="259200">
  <server>
    <address type="ipv4">64.86.88.117</address>
    <address type="ipv6">2001:05c0:8fff:ffff:8000:0001:c908:fe12</address>
  </server>
  <client>
    <address type="ipv4">201.8.254.18</address>
    <address type="ipv6">2001:05c0:8fff:ffff:8000:0000:c908:fe12</address>
    <keepalive interval="30">
      <address type="ipv6">2001:05c0:8fff:ffff:8000:0001:c908:fe12</address>
    </keepalive>
  </client>
</tunnel>
```

A saída do comando ***freenet6***, executado sem parâmetros em um terminal, causa a exibição do modo correto de utilização do *script*, e pode ser vista na Figura 13.

```
freenet6: parâmetros incorretos.
Modo de uso: freenet6 <command> | start [<verbose>] [-f conf_file] [-r secs]
  onde command pode ser:
   include   Adiciona os links de inicialização para '/etc/init.d/tspc'
   remove    Remove  os  links de inicialização para '/etc/init.d/tspc'
   stop      Para  a execução do TSP Client (o túnel IPv6-freenet6 é 'fechado')
   start     Inicia a execução do TSP Client (o túnel IPv6-freenet6 é 'aberto')
   <verbose> Nível de detalhe de  mensagens  exibidas  durante  o  processo  de
             inicialização do TSC Client. Possíveis valores: -v , -vv , -vvv
   -f        Lê o  arquivo  de  configuração  especificado,  ao invés do padrão
             (/etc/tsp/tspc.conf)
   -r        Tempo, em segundos, entre tentativas de conexão, até obter sucesso

Para outras opções referentes ao TSPC, execute "tspc -h" em um prompt shell.
```

Figura 13. Saída exibida na tela de um terminal, ao ser executado o comando ***freenet6***.

Os códigos dos *scripts* desenvolvidos são um pouco extensos, e por isso, estão anexados ao final deste projeto, nos Apêndices A e B. Nas subseções a seguir, descrevem-se as tecnologias envolvidas e justifica-se o porquê da sua utilização no projeto. Em seguida, veremos a ferramenta gráfica em ação.

6.4.1 A linguagem de programação Python

Python é uma linguagem de programação de alto nível, interpretada, interativa, orientada a objetos, e dinamicamente tipada, embora fortemente tipada (seus tipos são bem definidos, não há necessidade de coerção, e ainda assim os tipos de variáveis, parâmetros e retornos de funções são identificados pelo interpretador) [27][77][78]. Apesar de orientada a objetos, suporta outros paradigmas de programação, como a programação modular e a programação funcional.

Seu desenvolvimento teve início em 1990, no CWI (Instituto de Matemática e Ciência da Computação), em Amsterdã, na Holanda, comandado por Guido Van Rossum, sendo depois continuado pela *Python Software Foundation* (PSF). O nome Python teve a sua origem no grupo humorístico britânico Monty Python, criador do programa *Monty Python's Flying Circus*.

Um dos grandes atrativos da linguagem é a sua sintaxe, de fácil compreensão e rápida para ser dominada. Não há declaração de variáveis, e a delimitação dos blocos de instruções é feito por alinhamento ou indentação, isto é, não há delimitadores de blocos, como *Begin* e *End* de Algol, ou as chaves das linguagens C e JAVA. A construção dos tipos de dados é de alto-nível: estão presentes *strings*, dicionários, listas, tuplas, classes, dentre outros.

Python pode também ser extendida em módulos de compilação, como é o caso das linguagens C e C++. Os módulos de extensão podem definir novas funções e variáveis, bem como novos tipos de objetos. Em vários aspectos, assemelha-se a outras linguagens interpretadas conhecidas, como Tcl, Perl, Scheme e Ruby.

Sua utilização é crescente, e hoje vem sendo empregada em diversas áreas, como servidores de aplicação e computação gráfica, e por empresas de grande porte também, como Serpro e Embratel, no Brasil, por exemplo [78].

A linguagem Python foi escolhida para este projeto por possuir uma curva de aprendizagem muito menor do que a linguagem *shell script*. Além disso, o desempenho do interpretador da linguagem é bem satisfatório, permitindo a execução eficaz dos programas desenvolvidos.

6.4.2 O editor de interfaces gráficas Kommander

Uma das vantagens de sistemas *Unix-Like*, como é o caso do Linux, é a habilidade de fácil customização e automação de tarefas através da utilização de *scripts*. Entretanto, esses *scripts* geralmente necessitam funcionar em uma janela de terminal, e assim acabam não se integrando com os ambientes gráficos modernos utilizados atualmente. O Kommander [68] tem como intenção remover essa falta de integração, permitindo que os usuários criem facilmente aplicações gráficas e as utilizem em conjunto com alguma linguagem de *script* de sua escolha.

Kommander é uma aplicação voltada para ambientes KDE [65], construído com base na biblioteca de desenvolvimento Qt [65], e possui duas partes principais. A primeira parte é o editor, onde se constrói visualmente diálogos e aplicações e se editam os elementos de *scripts*. A segunda parte é um “lançador de aplicações”, uma espécie de interpretador mais conhecido como executor, e que processa os arquivos gerados pelo editor, permitindo sua execução. A Figura 14 mostra o ambiente de desenvolvimento do editor do Kommander.

É uma tecnologia que rompe com as idéias tradicionais. Não compila aplicações, não interfere no funcionamento e na configuração do sistema – que é tarefa dos *scripts* aos quais se

integra, e possui a capacidade de efetuar chamadas a funções DCOP – a linguagem de comunicação inter-processual do KDE, que são funções compiladas [68]. Dessa forma, produz diálogos e aplicações que são bem compactos para a transmissão na Internet, por exemplo, e de funcionamento bastante rápido, além de evitar questões relacionadas a binários de execução. Esses diálogos do Kommander integram-se facilmente às aplicações KDE, e o editor é cheio de *widgets* (componentes) do KDE. Também é possível desenvolver novos componentes.

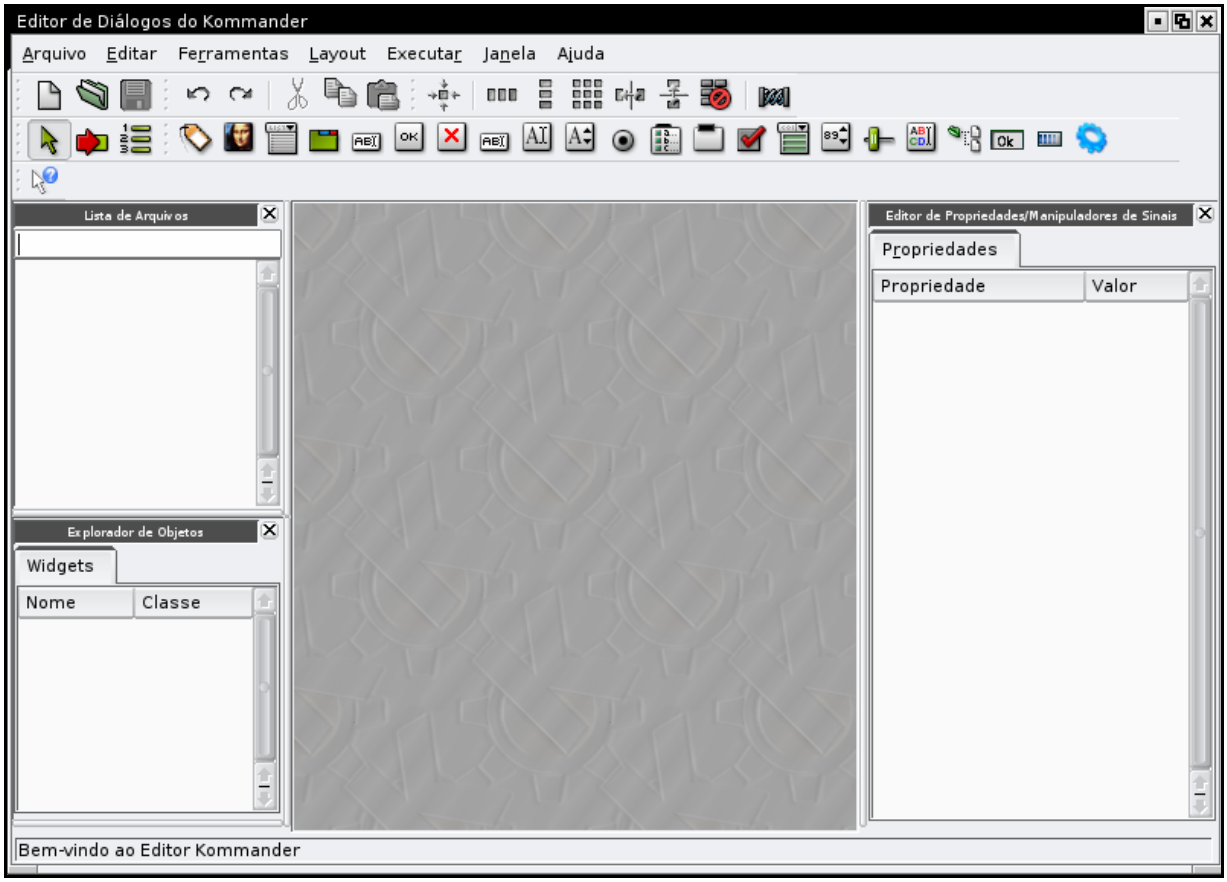

Figura 14. Tela do ambiente de trabalho do editor de diálogos Kommander.

A utilização do Kommander permite um modelo muito mais rápido de projeto e desenvolvimento para aplicações simples, por permitir que se pare de pensar demais na linguagem por trás da interface, revertendo à mais básica modelagem conceitual natural, e separando de forma eficaz a interface (*front-end*) dos arquivos de execução e da aplicação propriamente dita (*back-end*).

O Kommander foi escolhido para este projeto devido à sua capacidade de abstração do *back-end*, e à facilidade e rapidez de desenvolvimento. Além disso, é uma ferramenta específica para ambientes KDE, justamente o ambiente padrão adotado pelo 6ix Linux.

6.4.3 Ferramenta em execução

A aplicação pode ser acessada no ambiente de trabalho, através do menu da barra de tarefas, como mostra a Figura 15.a. No sub-menu IPv6 encontramos a ferramenta de configuração do acesso IPv6, a de configuração do *firewall*, além de outras.

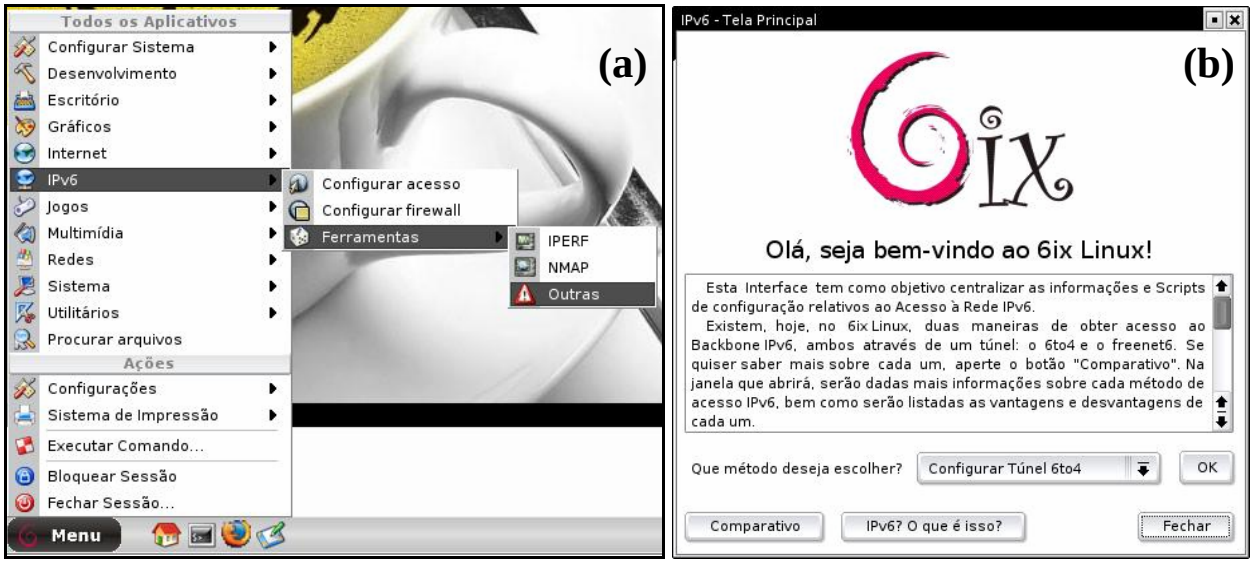

Figura 15. a) As ferramentas IPv6 podem ser acessadas através do sub-menu IPv6.
b) A tela principal da ferramenta de configuração do acesso IPv6.

A tela principal da aplicação mostra uma mensagem de boas-vindas, e um pequeno texto introdutório sobre a utilização da ferramenta (Figura 15.b). É nela também que escolhemos o método de acesso IPv6 a ser configurado, o *6to4* ou o *freenet6*. Se houver dúvida sobre qual método escolher, através do botão *Comparativo*, pode ser acessada uma tela com informações comparando as vantagens e desvantagens de cada um, em relação ao outro. Se o usuário não está muito familiarizado com o IPv6, se nunca ouviu falar, ou não sabe nem o que é o IP, pode acessar uma tela de explicação sobre o protocolo e alguns conceitos básicos relacionados, através do botão *IPv6? O que é isso?*. Também há a indicação de alguns sítios da Internet falando a respeito do assunto, servindo de ponto de partida para aqueles que tiverem maior curiosidade ou quiserem se aprofundar no assunto. Essas duas telas de ajuda são mostradas na Figura 16.

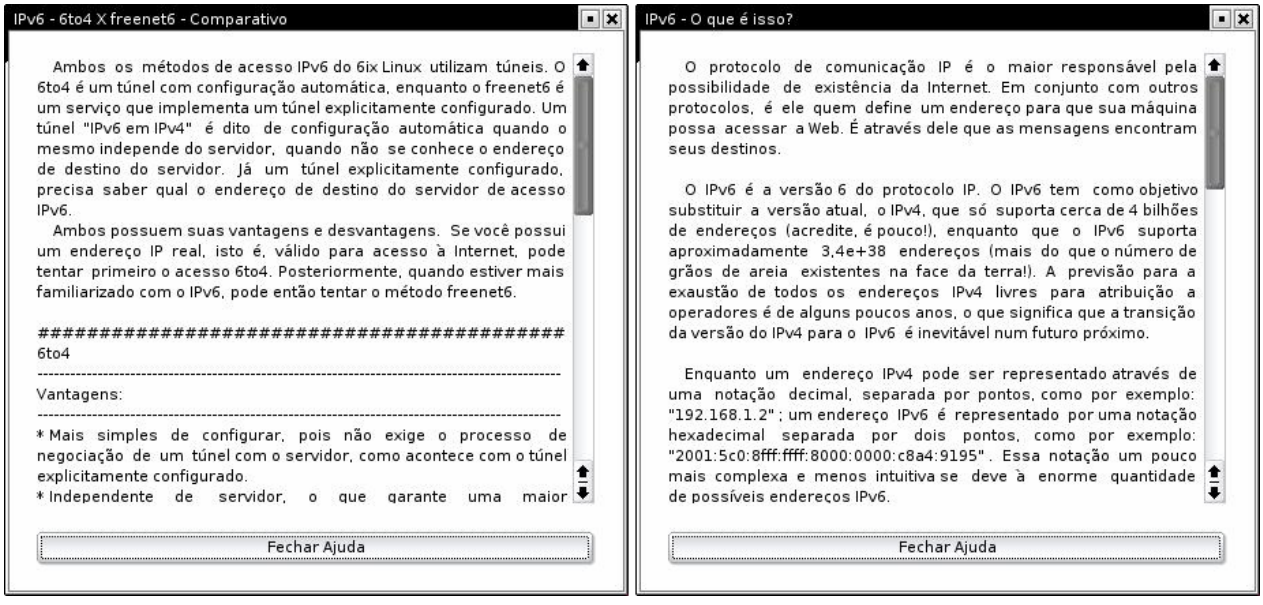

Figura 16. Telas de ajuda da janela principal da aplicação de configuração de acesso IPv6.

Se o método escolhido for o tunelamento *6to4*, a tela de configuração dessa ferramenta será apresentada (Figura 17). Nesta tela, é possível configurar ou desfazer as configurações de um túnel *6to4*, desligar a interface, parando o serviço, ou ligar a interface do túnel, inicializando o serviço. É possível ainda listar as interfaces ativas no sistema, e uma tela de ajuda pode ser exibida, caso existam dúvidas na operação da ferramenta. O usuário deve saber a interface que está conectada à Internet, e pode escolher se o túnel configurado será ligado automaticamente durante as próximas inicializações do sistema, ou não. Caso a janela seja fechada através do botão *Fechar*, será exibida novamente a tela principal da aplicação.

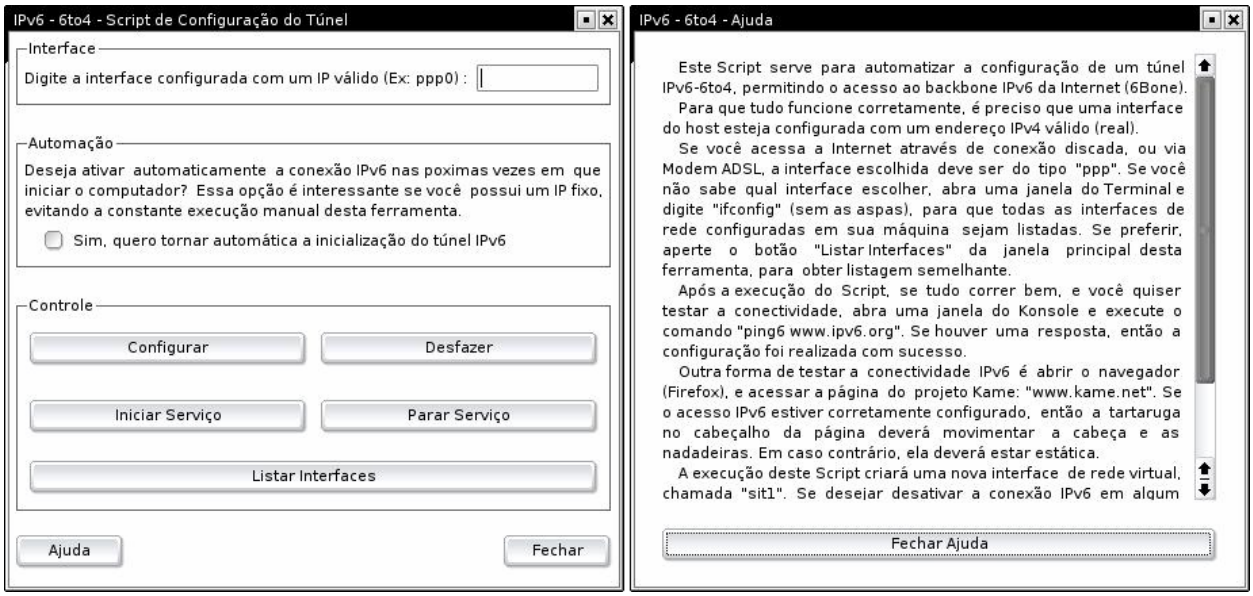

Figura 17. Telas de configuração do túnel 6to4 e de ajuda de utilização da ferramenta.

Caso o método escolhido seja o tunelamento *freenet6*, será exibida a tela de configuração correspondente **(**Figura 18). Nesta tela, é possível escolher o nível de detalhes de informações exibidos durante a configuração, indicar um arquivo de configuração diferente do padrão, contendo novas informações, e até mesmo o tempo a aguardar entre tentativas de configuração do túnel. Assim como para os túneis *6to4*, aqui podemos iniciar ou parar o serviço, e adicioná-lo à inicialização, ligando o túnel automaticamente nas próximas vezes em que for efetuado o *boot* da máquina, ou remover essa configuração da inicialização. Caso a janela seja fechada através do botão *Fechar*, será exibida novamente a tela principal da aplicação.

Para auxiliar a compreensão do fluxo entre as telas mais importantes da ferramenta, apresentamos, na Figura 19, um fluxograma simples.

Após o desenvolvimento da ferramenta de configuração de acesso IPv6, foi criada uma nova versão da distribuição, e uma nova imagem do sistema foi gravada em CD, seguindo os moldes da primeira versão. A essa versão do 6ix Linux, chamamos de *segunda versão estável*.

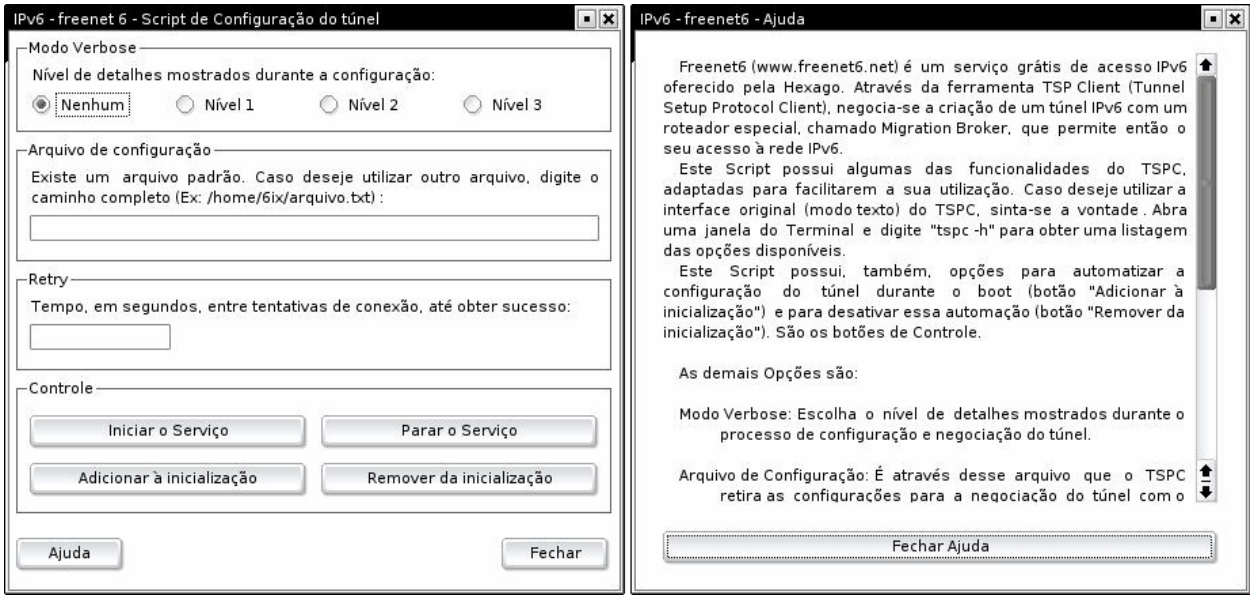

Figura 18. Telas de configuração do túnel *freenet6* e de ajuda de utilização da ferramenta.

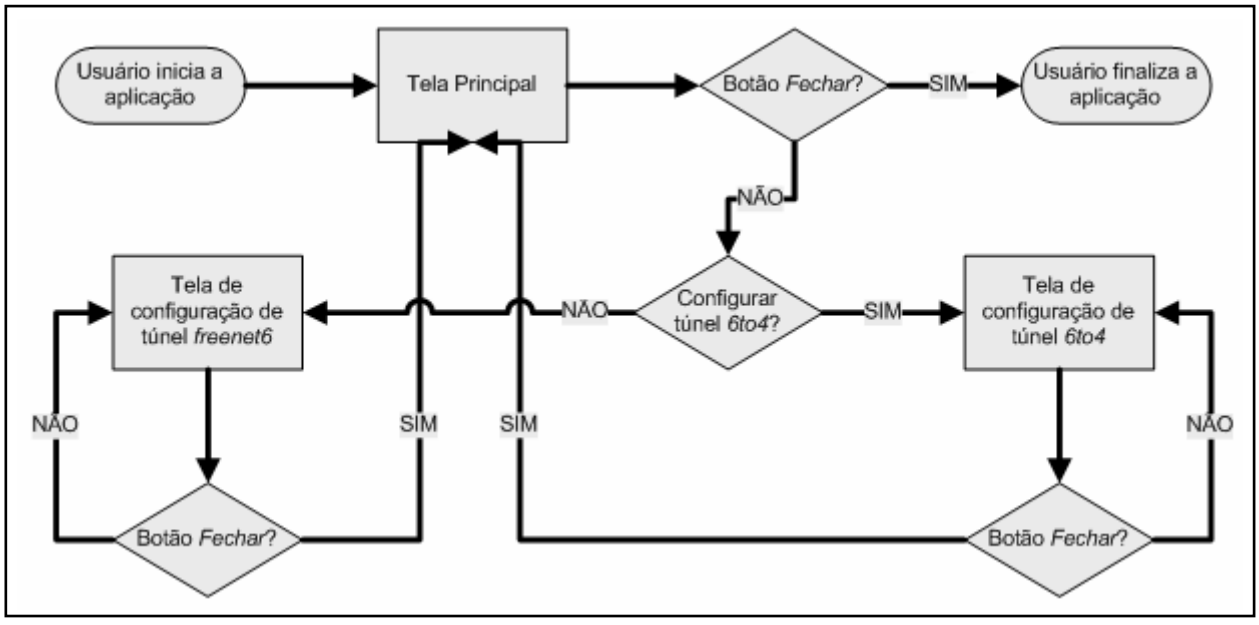

Figura 19. Fluxograma de apresentação das telas mais importantes da ferramenta.

6.5 Ferramenta de configuração do firewall

Como discutido anteriormente, o IPv6 possui características incorporadas que provêem maior nível de segurança do que o IPv4. Mas ainda assim, não resolve todas as questões de segurança, e nem exclui a necessidade de ferramentas como IDSs e *firewalls*, que ajudam a combater ataques e invasões [9] [34] [35].

Um *firewall* é um dispositivo de rede cuja finalidade é restringir o tráfego de mensagens enviadas de uma rede privada para a Internet, e vice-versa, dificultando a invasão da rede, e protegendo-a contra ataques como os de negação de serviço ou propagação de vírus, por

exemplo. Um *firewall* pode ainda realizar tarefas semelhantes às de um roteador, como a tradução de endereços (NAT) ou o redirecionamento de pacotes.

Para o Linux, existem diversos programas que se propõem a exercer a função de *firewall*. Dentre todos, destaca-se o *iptables*, o mais conhecido, e uma das mais completas, estáveis e eficientes ferramentas, para a plataforma [23][73]. Tanto é que na maioria das distribuições linux, já está presente, compilado e integrado ao Kernel.

O *iptables* é um *firewall* de nível de pacotes e de arquitetura modular, em que novas funcionalidades podem ser acrescentadas à medida do necessário, aumentando o poder de análise de pacotes.

A estrutura do *iptables* é composta por tabelas, cadeias e regras. As tabelas armazenam as cadeias, que, por sua vez, são compostas por regras. Através das regras, efetua-se a filtragem de pacotes, que consiste em analisar o cabeçalho de cada pacote que pretende atravessar as fronteiras da rede interna, permitindo ou não o encaminhamento desses pacotes. Essa filtragem pode ser baseada em vários fatores, como no protocolo, ou no endereço de origem ou de destino. No entanto, por ser um *firewall* de nível de pacote, por padrão o *iptables* só consegue examinar protocolos de camada de transporte ou de rede.

O 6ix Linux possui o *iptables* instalado, e uma ferramenta gráfica para facilitar a configuração do *firewall* – o que para usuários leigos não é uma tarefa fácil. Basicamente, o que a ferramenta faz é auxiliar na criação de um *firewall* de regras simples, chamado de *firewall* de bloqueio, em que todas as portas de entrada, TCP e UDP, são bloqueadas. Há ainda a opção de abrir determinadas portas, escolhidas pelo usuário. A Figura 20 mostra a sua interface, quando em execução. Uma tela de ajuda também foi desenvolvida, para sanar dúvidas eventuais sobre a utilização dessa ferramenta. Esta tela também está mostrada na Figura 20.

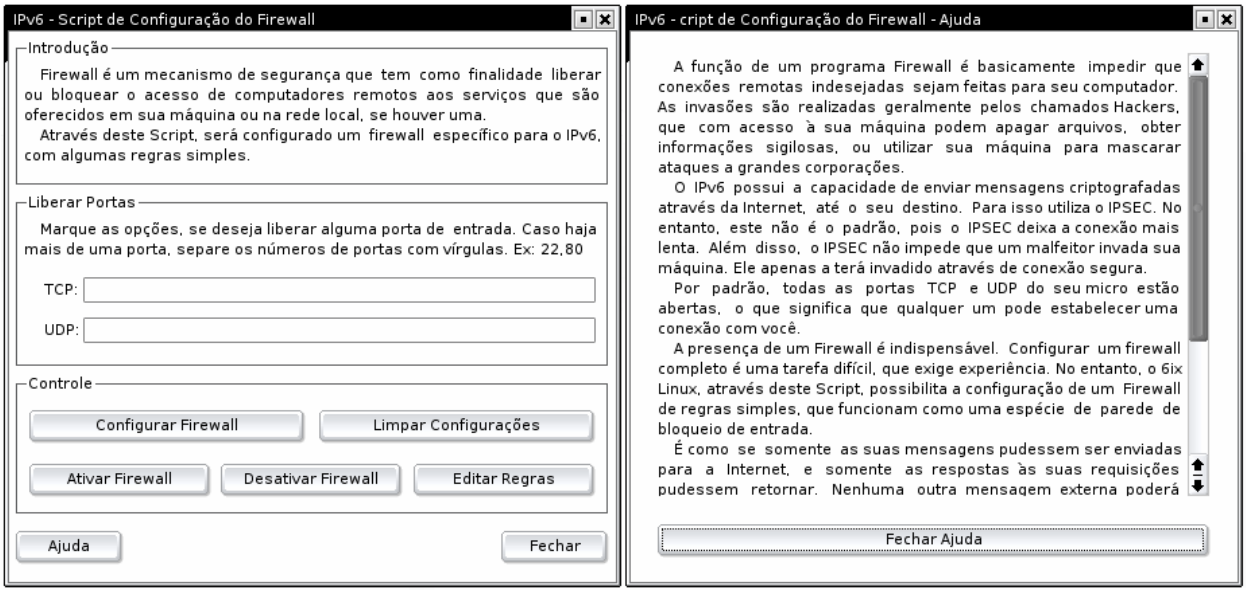

Figura 20. Telas da ferramenta de configuração de firewall IPv6 do 6ix Linux.

Assim como a ferramenta de configuração do acesso IPv6, a ferramenta de configuração do *firewall* específico para IPv6 também foi desenvolvida em Kommander, juntamente com a linguagem Python. O *script* Python é quem realiza toda a configuração do *firewall*. Ao Kommander, cabe a apresentação visual da ferramenta.

O *script* possui várias funções, como mostra a saída de tela na Figura 21. Essa tela é exibida quando o comando ***6ixfirewall*** é executado, em modo texto, no terminal, com algum parâmetro incorreto.

```
6ixfirewall: parâmetros incorretos.
Modo de uso: 6ixfirewall [-t <tcp>] [-u <udp>] | <clear> | <stop> | <start>
  onde:
   <tcp>    Lista de portas TCP de entrada que deverão ser abertas
            (separadas por vírgulas)
   <udp>    Lista de portas UDP de entrada que deverão ser abertas
            (separadas por vírgulas)
   <clear>  Apaga as alterações feitas no arquivo '/etc/6ixfirewall.sh'
   <stop>   Desativa o firewall
   <start>  Ativa o firewall
```

Figura 21. Tela mostrando o modo de utilização correto do *script*, em modo texto.

O *firewall iptables* é acionado assim que o sistema operacional é carregado. No entanto, nesse momento não existem ainda regras configuradas, e portanto o *firewall* não analisa os pacotes, deixando todas as portas de comunicação vulneráveis. O módulo do *iptables* específico para a configuração de regras envolvendo o protocolo IPv6 é o *ip6tables* [2]. A ativação do *firewall* se dá efetivamente através do comando ***6ixfirewall start***, que executa o arquivo */etc/6ixfirewall.sh*, previamente configurado através do comando ***6ixfirewall –t <tcp> -u <udp>***. Seguindo o mesmo raciocínio, desativar o *firewall* consiste em apenas remover as regras da lista de execução do *iptables*. É o que a opção ***6ixfirewall stop*** faz, executando o comando ***ip6tables -F***. O comando ***6ixfirewall clear*** torna o arquivo */etc/6ixfirewall.sh* à configuração original, sem nenhuma regra para ser acrescentada ao *iptables*.

As regras adicionadas ao *firewall* pelo *script* são simples. Primeiro, vem uma regra que limpa qualquer regra em execução. Depois, vêm as regras responsáveis pelo fechamento das portas TCP e UDP. Por último, vêm as regras responsáveis pela abertura de portas TCP e UDP específicas, se houver alguma. Abaixo, no Quadro 5, vemos as regras adicionadas pelo *script*. No exemplo, apenas uma porta TCP de entrada é aberta – a porta 80, e todas as portas UDP permanecem fechadas. O *iptables* lê as regras de cima para baixo, e se um pacote se enquadra nas restrições de uma regra, as outras mais abaixo não são lidas. Portanto, as regras do topo da lista têm prioridade sobre as demais regras.

Quadro 5. Exemplo de regras adicionadas pelo *script* de configuração de *firewall* do 6ix Linux.

```
ip6tables -F
ip6tables -I INPUT -i sit+ -p tcp --syn -j DROP
ip6tables -I INPUT -i sit+ -p udp -j DROP
ip6tables -I INPUT -i sit+ -p tcp --dport 80 -j ACCEPT
```

A opção *-F* do *ip6tables* serve para apagar todas regras em execução, enquanto a opção *-I* serve para inserir uma regra no topo da lista de regras. Sendo assim, as últimas três regras do quadro acima são inseridas na lista de regras em ordem contrária. A cadeia *INPUT* significa que os quadros são avaliados quando são recebidos pelo *firewall* e se destinam a ele mesmo. O parâmetro *-i* especifica o tipo de interface avaliada pela regra, no caso *sit+*, que significa todas as interfaces do tipo *sit*, que é o tipo de interface para IPv6 tunelado. Já o parâmetro *-p* serve para definir o protocolo analisado, geralmente *tcp*, *udp* ou *icmp*.

A segunda regra de cima para baixo, mostrada no quadro, bloqueia a entrada de pacotes TCP do tipo SYN. Esses pacotes são utilizados no processo de negociação e estabelecimento de comunicação do TCP, o *3-way handshake*. Se esse tipo de pacote não tem permissão para acesso

ao *firewall*, significa que conexões TCP externas não serão estabelecidas. Já a terceira regra, proíbe a entrada de quaisquer conexões que utilizem o protocolo UDP. A quarta e última regra do exemplo é responsável pela abertura da porta TCP de entrada de número 80. A extensão de regra *--dport* é utilizada para informar que a porta liberada é a de destino do pacote que chega, ou seja, a porta 80 do *firewall*.

Caso o *firewall iptables* esteja habilitado e configurado para controlar as conexões IPv4, será preciso liberar o tráfego IPv6 através da interface tunelada. Se a conexão à Internet é feita através da interface *ppp0*, por exemplo, então a interface de túnel virtual, *sit1*, utiliza essa interface para realizar a comunicação tunelada. Para liberar o tráfego, neste caso, as regras ficariam como as mostradas no quadro abaixo. A opção *-o* é semelhante à opção *–i*, mas para interfaces de saída.

Quadro 6. Regras adicionadas ao *iptables* para liberar o tráfego de pacotes IPv6.

```
iptables -I INPUT -i ppp0 -p ipv6 -j ACCEPT
iptables -I OUTPUT -o ppp0 -p ipv6 -j ACCEPT
```

O código-fonte do *script* Python pode ser conferido no Anexo C deste trabalho. O arquivo do *script* foi colocado na pasta */usr/local/bin*. Já os arquivos de interface gráfica, estão localizados no diretório */usr/share/IPv6*.

Após nova avaliação do comportamento do sistema, já com a ferramenta de configuração do *firewall* acrescentada à distribuição, novamente geramos uma nova imagem, logo em seguida gravando-a em um CD. A essa versão, foi dado o nome de *terceira versão estável*.

6.6 Instalação de ferramentas de configuração de ambiente e rede local IPv6, e utilitários

O propósito principal da distribuição 6ix Linux é permitir a configuração do acesso à Internet IPv6, de forma simples e visual, para todos os níveis de usuários. Entretanto, além da ferramenta de configuração do acesso, com sua interface e seus *scripts*, também outras funcionalidades, específicas para a configuração de ambientes de rede local IPv6, foram incluídas, além de utilitários com diversos fins. Existem sítios na *web* que listam várias ferramentas com suporte ao IPv6 já implementado [49][60].

Esta fase de produção consistiu basicamente na instalação de vários pacotes, em formato *.deb*, referentes a cada programa escolhido para utilização no 6ix Linux. Pacotes *.deb* são os pacotes de instalação das distribuições Debian e suas derivadas [29]. São os pacotes que estão disponíveis nos repositórios da ferramenta *apt-get*, mas também podem ser salvos localmente e instalados de forma manual através da ferramenta *dpkg*.

A vantagem de se instalar pacotes *.deb* é que o programa já vem compilado para determinada arquitetura, e os arquivos são postos automaticamente nos locais adequados para permitir uma melhor execução do programa. Além disso, algumas ações pré-instalação e pós-instalação também podem ser configuradas, para auxiliar no processo de instalação desses pacotes.

A seguir, são listados os *softwares* instalados, detalhados individualmente nas próximas subseções. A última subseção relata as dificuldades encontradas durante o desdobramento desta fase. Os pacotes instalados foram:

- **apache2:** Servidor Web;
- **bind9:** Servidor DNS;
- **freeswan:** IPSEC;
- **iperf** e **jperf:** Análise de desempenho da rede;
- **netcat6:** Exploração e depuração da rede;
- **nmap** e **nmapfe:** Auditoria de segurança;
- **quagga:** Suíte de roteamento;
- **radvd:** Configuração de endereços;
- **tcpdump** e **ethereal:** Analisadores de tráfego de pacotes.

6.6.1 Servidor Apache

O Apache é o servidor de aplicações *Web* mais conhecido e mais utilizado em todo o mundo. Parte integrante e principal projeto da *Apache Software Foundation*, o Apache [44] é reconhecido pela sua grande estabilidade e ferramentas variadas para dar suporte a praticamente todo tipo de requisição *Web*, sendo capaz de trabalhar perfeitamente com diversos protocolos, como HTTP, FTP, SSL, PHP (*Personal Home Pages*), dentre outros. Através do Apache é possível disponibilizar páginas escritas em HTML (*HyperText Markup Language*) na Internet, ou mesmo *sites* – ou sítios – inteiros.

O Apache é um projeto de esforço de desenvolvimento colaborativo com objetivo de desenvolver um servidor *web* com características como portabilidade, alto desempenho, robustez, documentação completa, flexibilidade e com padrões de excelência e qualidade. Tem em seu grupo de trabalho programadores das Universidades MIT, Berkeley, Stanford, e empresas como IBM, Sun, HP, Compaq, RedHat entre diversas outras.

É uma ferramenta de código aberto, distribuída livremente, através de licença própria irrestrita, compatível com a licença GPL. Trabalha com a especificação HTTP/1.1, e permite mudanças em suas características mesmo em suas partes mais internas, através da utilização de módulos. Tem sua própria API (*Application Programming Interface*), facilitando e padronizando toda a programação.

Atualmente em sua versão 2.0, o Apache já vem com suporte ao protocolo IPv6 ativado por padrão. Se existir alguma conexão IPv6 na máquina, o servidor atenderá automaticamente às requisições feitas através do protocolo [10].

6.6.2 Bind

O Bind (*Berkeley Internet Name Domain*) é um dos mais conhecidos, robustos e utilizados servidores DNS para a plataforma linux [46]. Assim como o Apache, possui uma API própria e é distribuído de forma totalmente gratuita.

O suporte do Bind ao IPv6 é garantido através da utilização de registros do tipo AAAA, ou quad-A. Novos tipos de registros específicos para IPv6, como o A6 e o DNAME, estão ainda em fase de desenvolvimento, e por isso não são suportados ainda.

O processo de configuração das zonas e domínios, para IPv6, é semelhante ao processo utilizado para configurar domínios IPv4, não havendo grandes dificuldades para ser configurado por pessoas que já possuam experiência prévia com o Bind [1].

6.6.3 FreeS/WAN

O *Free Secure Wide Area Network* (FreeS/WAN) [53] é uma implementação livre da estrutura IPSEC, específica para o linux, possibilitando a realização de comunicações fim-a-fim seguras, utilizando pseudo-túneis através da Internet, chamados geralmente de VPNs.

O suporte do FreeS/WAN ao IPv6 ainda se encontra em estágio experimental. Infelizmente o projeto não está mais em atividade, mas outros projetos, como o OpenS/WAN e o StrongS/WAN basearam-se no código da ferramenta para dar continuidade ao seu desenvolvimento. Além disso, os próprios mantenedores do Kernel Linux vêm trabalhando para dar suporte de segurança ao IPv6. Assim, não deve demorar demais até que o IPv6 adquira finalmente uma implementação IPSEC completa.

6.6.4 Iperf e Jperf

Iperf é uma ferramenta desenvolvida pelo *National Laboratory for Applied Network Research* (NLANR) [74], uma subdivisão especial da *National Science Foundation*, órgão americano que participou na criação da Internet.

O Iperf é uma ferramenta utilizada para medir o desempenho de conexões TCP e UDP [58]. Possui um grande número de opções, sendo capaz de medir parâmetros como largura de banda, atrasos de propagação, *jitter*, e taxa de perda de pacotes. O Iperf também suporta a mensuração de medidas através do protocolo IPv6.

Jperf é uma interface gráfica de usuário, um *front-end* para o Iperf desenvolvido em JAVA, e que é capaz de produzir gráficos baseados no resultado das execuções, facilitando a análise dos dados.

6.6.5 Netcat6

Netcat6 [72] é uma ferramenta surgida a partir da reescrita de uma outra ferramenta, Netcat, objetivando o aperfeiçoamento de suas características e a adição do suporte total ao protocolo Ipv6.

É uma ferramenta versátil, capaz de enviar e receber dados através de conexões TCP e UDP, sendo geralmente utilizada para investigação, exploração e depuração do funcionamento de uma rede. É um utilitário bastante conhecido no meio *Hacker*, por poder assumir as mais diversas funções, indo de um simples Telnet até a geração de ataques de força bruta ou a procura por portas remotas abertas (*portscanning*).

6.6.6 Nmap e NmapFE

O Nmap (*Network Mapper*) [75] é um escaneador de *hosts* que usa recursos avançados para verificar o estado do seu alvo. A ferramenta é gratuita e disponibilizada para várias plataformas, inclusive Linux. É uma ferramenta de código aberto utilizada para realizar auditoria de segurança e explorar toda uma rede.

Desenvolvida com capacidade para agir eficientemente em redes de grande porte, também pode ser usada satisfatoriamente para testar uma única máquina. Utiliza pacotes IP de várias maneiras diferentes para determinar que *hosts* estão disponíveis na rede, que serviços são oferecidos – citando nome e versão de cada serviço, e que sistema operacional e versão do sistema operacional cada máquina utiliza. Verifica ainda a existência de *firewalls*, dentre outras várias opções.

É uma ferramenta muito famosa, conhecida por características como portabilidade, flexibilidade, facilidade de uso, boa documentação e grande poder. Devido à flexibilidade e

eficiência, o Nmap pode ser considerado tanto uma ferramenta *Hacker* como um excelente utilitário para consultores de segurança e administradores de rede.

A versão do NMap para linux possui ainda uma ferramenta chamada NMapFE (*NMap Front-End*), que nada mais é do que uma interface gráfica para o Nmap.

O suporte ao protocolo IPv6 ainda não está completamente implantado, mas as principais funções já são capazes de examinar conexões através do novo protocolo.

6.6.7 Quagga

É uma ferramenta livre que permite o gerenciamento de protocolos de roteamento baseados na arquitetura TCP/IP, em ambientes linux, de forma centralizada. O Quagga é uma continuação de outra ferramenta, a Zebra, que praticamente não tem sido mais desenvolvida.

O Quagga é composto por uma suíte de protocolos de roteamento [79]. São implementados: OSPFv2, OSPFv3, RIP v1, RIPv2, RIPv3 e BGPv4+.

Através da ferramenta, o sistema se comporta de forma semelhante a um roteador, no que diz respeito à configuração através de terminal, com comandos que automatizam tarefas e abstraem as modificações exercidas no sistema.

O IPv6 está presente, suportado pelos protocolos OSPFv3, RIPv3 e BGPv4+.

6.6.8 RADVD

O Serviço de Anunciação de Rotas, RADVD (*Router Advertisement Daemon*) permite a um sistema linux agir como o distribuidor de endereços IPv6 para uma rede local, no processo de configuração automática de endereços dito sem estado (*stateless*), fazendo as vezes de um roteador [80].

É um serviço desenvolvido especificamente para auxiliar o protocolo IPv6, e não tem aplicabilidade em redes IPv4.

6.6.9 TCPDump e Ethereal

São analisadores de tráfego de pacotes, de código livre, usados por profissionais para analisar, investigar e resolver problemas eventuais em uma rede, ou até mesmo para acompanhamento do tráfego e aprendizagem do funcionamento de protocolos específicos.

O TCPDump é um utilitário de modo texto, responsável pela captura e armazenagem de informações acerca dos pacotes [82]. O Ethereal é uma ferramenta gráfica que, baseada no TCPDump, disponibiliza um grande número de opções para analisar de forma aprofundada o conteúdo de pacotes [50].

A quantidade de protocolos suportados pela dupla é enorme, praticamente abrangendo todos os padrões da Internet e quase todos os protocolos privados mais conhecidos. O IPv6 e o ICMPv6, é claro, fazem parte desta lista.

6.6.10 Dificuldades de instalação

O processo de remasterização de uma distribuição não é uma tarefa tão simples, quanto pode parecer à primeira vista. Quando são necessárias alterações de funcionamento mais profundas, e em grande quantidade, diversos percalços aparecem durante a modificação do sistema operacional original.

Das dificuldades encontradas durante o desenvolvimento do 6ix Linux, as que se fizeram presentes específicamente durante esta fase de instalação de ferramentas merecem um destaque maior, por ter levado um tempo considerável para encontrar uma solução.

A primeira tentativa para a instalação dos utilitários e ferramentas IPv6, foi através da ferramenta *apt-get*. Por algum motivo, somente uns poucos pacotes puderam ser instalados. Para os demais, durante a checagem de dependências, antes de prosseguir com a instalação, era exibida uma mensagem semelhante à mostrada na Figura 22. Essa mensagem alertava sobre a periculosidade da operação, que poderia tornar o sistema instável, e até inutilizá-lo. Logicamente, uma mensagem como essa não deveria aparecer durante a instalação de programas tão comuns e conhecidos como o Apache ou o Nmap, deixando claro que algo não estava correto.

```
6ix@6ix:/root# sudo apt-get install mozilla-firefox
Lendo Lista de Pacotes... Pronto
Construindo Árvore de Dependências... Pronto
Os pacotes extra a seguir serão instalados:
gcc-4.0-base libatk1.0-0 libc6 libc6-dev libgcc1 libglib2.0-0 libglib2.0-data libpango1.0-0 libpango1.0-common
libstdc++6 libxinerama1 libxrender1 linux-wlan-ng locales
Pacotes sugeridos :
glibc-doc manpages-dev ttf-kochi-gothic ttf-kochi-mincho ttf-thryomanes ttf-baekmuk ttf-arphic-gbsn00lp
ttf-arphic-bsmi00lp ttf-arphic-gkai00mp ttf-arphic-bkai00mp mozilla-firefox-gnome-support latex-xft-fonts
xprint
Pacotes recomendados :
libatk1.0-data gcc c-compiler linux-wlan-ng-doc linux-wlan-ng-0.2.2-modules apt-src
linux-wlan-ng-0.2.1-modules
Os pacotes a seguir serão REMOVIDOS:
console-common console-data console-tools e2fsprogs initrd-tools modutils nfs-common nfs-kernel-server
sysvinit
Os NOVOS pacotes a seguir serão instalados:
gcc-4.0-base libstdc++6 libxinerama1 mozilla-firefox
Os pacotes a seguir serão atualizados :
libatk1.0-0 libc6 libc6-dev libgcc1 libglib2.0-0 libglib2.0-data libpango1.0-0 libpango1.0-common libxrender1
linux-wlan-ng locales
AVISO: Os pacotes essenciais a seguir serão removidos
Isso NÃO deve ser feito a menos que você saiba exatamente o que está fazendo!
e2fsprogs sysvinit
11 pacotes atualizados, 4 pacotes novos instalados, 9 a serem removidos e 622 não atualizados.
É preciso fazer o download de 21,3MB de arquivos.
Depois de desempacotamento, 17,6MB adicionais de espaço em disco serão usados.
Você está prestes a fazer algo potencialmente destruidor
Para continuar digite a frase 'Sim, faça o que eu digo!'
```

Figura 22. Alerta mostrado durante a instalação de pacotes, através da ferramenta *apt-get*.

A solução encontrada para esses casos foi efetuar o *download* dos pacotes desejados, e tentar instalá-los localmente através da ferramenta *dpkg* [29], na esperança de que o erro fosse decorrente do *apt-get*. Novamente, houve uma divisão. Alguns pacotes foram instalados, enquanto outros apresentaram erro. Esse erro apresentado foi comum em todos os casos de insucesso, e consistia basicamente num problema de versão da biblioteca *libc6*, um pacote básico do sistema utilizado por todos os programas para controle de hora e data.

A versão requerida era a *libc6 v2.3.22* ou superior. Constatou-se que a versão presente no sistema 6ix Linux era a *libc6 v.2.3.21*. Após tentar atualizar a versão da biblioteca, houve falha através do *apt-get*, apresentando mensagem semelhante às demais, e através da ferramenta *dpkg*, que exigia a instalação de um grande número de novos pacotes. O jeito, então, foi assumir que devido à pequena diferença entre versões da biblioteca, os programas instalados deveriam funcionar de forma correta. Bastava apenas mudar a versão requerida nos pacotes, para a versão presente no sistema.

As versões de pacotes dos quais depende um pacote *.deb* são definidas durante a criação do pacote. Esses pacotes *.deb*, ou pacotes DEBIAN, nada mais são do que um conjunto de

arquivos compactados, juntamente com as informações de locais adequados de instalação para cada um, e de algumas instruções de controle, como por exemplo as versões das dependências, e instruções de execução pré-instalação e pós-instalação dos arquivos.

Assim, foi preciso desempacotar os pacotes DEBIAN problemáticos, modificar a versão do pacote exigido *libc6 v2.3.22* para a *libc6 v2.3.21*, e recriar os pacotes *.deb*. Não houve problemas nesta etapa, e em seguida passou-se à tentativa de instalação dos novos pacotes, através da feramenta *dpkg*. Os programas foram instalados com sucesso, e seu funcionamento não apresentou nenhum comportamento inadequado, levando à conclusão de que a instalação finalmente havia sido executada por completo, e com sucesso.

Posteriormente, após a geração da *quarta versão estável*, foi descoberto o motivo do erro ocorrido na utilização do *apt-get*. A configuração do arquivo */etc/apt/source.list*, utilizado para indicar os repositórios de pacotes usados pelo *apt-get*, estava incorreta. Alguns dos repositórios estavam entrando em conflito com outros. Devido à essa má-configuração do arquivo, estavam ocorrendo confusões entre as versões dos programas a serem instalados. Assim, foi efetuada a correção do arquivo, e a versão da biblioteca do sistema foi atualizada. Os programas instalados não precisaram de nenhuma modificação, e foi gerada então, uma nova e última versão estável do sistema, a *versão estável final do 6ix Linux release 1.0*.

6.7 Experimentos

O método de acesso à rede IPv6 disponível no 6ix Linux é o tunelamento. A intenção da distribuição é servir como intensificador e distribuidor de conhecimento acerca do novo protocolo da Internet, devendo ser utilizado principalmente em redes e computadores domésticos ou em instituições de pequeno porte, os SOHOs (*Small Offices/Home Offices*), que ainda não possuem provedores de acesso com suporte ao IPv6, e nem terão tão cedo.

Sem provedores adequados, tais redes ficam impossibilitadas de utilizar o método de pilha dupla para obter acesso ao backbone IPv6, o que não acontece com o tunelamento, que contorna essa situação justamente utilizando a rede IPv4 para alcançar a Internet IPv6, mostrando-se totalmente adequado para os fins a que se destina o 6ix Linux.

O tunelamento de pacotes IPv6-em-IPv4 pode ser realizado através de configuração explícita ou automática, como já foi discutido, e o 6ix Linux disponibiliza um representante de cada categoria, permitindo a configuração de um túnel automático *6to4*, ou de um túnel *freenet6* configurado explícitamente.

Com o intuito de aprofundar conhecimentos sobre o funcionamento prático de cada tipo de túnel, e de comparar o desempenho de cada um, foram realizados alguns testes simples, utilizando as ferramentas Ping, Traceroute e Tracepath [11][32] para demonstrar na prática a capacidade de comunicação IPv6 da distribuição 6ix Linux, e servir como base para elencar e classificar as vantagens e desvantagens de cada tipo de túnel em relação ao outro.

As ferramentas supracitadas, utilizadas no experimento, são bastante conhecidas e utilizadas no dia-a-dia por administradores de rede para aferir a qualidade e o desempenho de LANs, WANs e até do roteamento através da Internet, estando disponíveis em praticamente todo sistema operacional.

Como os testes foram realizados através do protocolo IPv6, foi necessário utilizar as versões específicas das ferramentas, para trabalhar com o protocolo. Esses utilitários, *ping6*, *traceroute6* e *tracepath6*, pertencentes ao pacote *iputils*, são equivalentes às ferramentas citadas à pouco, distinguindo-se apenas por possuir a capacidade de lidar com o protocolo IPv6, e por utilizarem pacotes ICMPv6 ao invés de ICMP. O funcionamento é praticamente idêntico, mudando apenas os formatos dos endereços.

A seguir, descrevem-se as ferramentas utilizadas, e em seguida os experimentos realizados, e seus resultados.

6.7.1 A ferramenta Ping

A ferramenta Ping utiliza mensagens ICMP do tipo *echo request* e *echo reply* para testar a capacidade da máquina local de se comunicar com outras máquinas, remotas, inclusive medindo o tempo de ida e volta de cada pacote até o seu destino, o RTT (*Round Trip Time*). O comando Ping permite ainda configurar o tamanho e a quantidade dos pacotes a serem enviados até um destinatário, e a definição de outras características como o tempo de espera para cada pacote ou o número máximo de saltos a serem percorridos por cada um.

Os pacotes são enviados ordenadamente pela ferramenta Ping, e para cada pacote enviado e recebido de volta, é exibido na tela um conjunto de informações como o número de sequência e o valor do campo TTL desse pacote. Ao final da execução, a ferramenta apresenta um resumo, contendo informações como o tempo de duração do teste, a taxa de perda de pacotes, e os valores mínimo, médio e máximo para o RTT. As medidas de tempo da ferramenta são praticamente todas em milissegundos (*ms*).

O menor valor de um pacote ICMP de Ping é o equivalente ao tamanho do cabeçalho, 8 *bytes*. No entanto, somente a partir de 16 *bytes* é possível obter estatísticas sobre o RTT de cada pacote, pois além dos 8 *bytes* do cabeçalho, outros 8 *bytes* são utilizados para passar informações sobre a hora em que o pacote foi enviado, permitindo o cálculo do tempo de ida e volta. O padrão do tamanho de pacotes Ping para sistemas unix, é de 64 *bytes*. Além dos 16 *bytes*, o restante do pacote é preenchido com *bits* aleatórios, até alcançar o tamanho requisitado. O tamanho dos pacotes é importante, pois influencia nos tempos de resposta, e na quantidade de pacotes descartados. Outros valores padrões do Ping em sistemas unix são o TTL de 60 saltos e o tempo de intervalo entre envio de pacotes, de 1 segundo.

O Ping é uma ferramenta de uso simples, que pode ser utilizada para testar a capacidade de comunicação em rede de um *host*. No entanto, não é a ferramenta mais adequada para medir o desempenho de uma rede, pois pacotes ICMP possuem baixa prioridade de processamento em roteadores, o que causa maiores valores no tempo de ida e volta dos pacotes, se houver congestionamentos durante o caminho.

Outro problema na utilização do Ping é que se tornou comum a sua utilização em ataques de negação de serviço como o *Ping da Morte*, em que uma elevada quantidade de pacotes de grande tamanho são enviados em curtos intervalos de tempo a um destinatário, sobrecarregando-o e causando a sua incapacidade de comunicação. Dessa forma, a maioria das instituições adotou um conjunto de regras de *firewall* para prevenir esse tipo de ataque, impossibilitando a utilização da ferramentas Ping.

6.7.2 A ferramenta Traceroute

O aplicativo Traceroute é utilizado para investigação mais aprofundada do comportamento dos roteadores envolvidos em uma comunicação. Com o Traceroute, é possível descobrir o caminho percorrido pelos pacotes desde o remetente até o destino final. Pode ser usado também para investigação de falhas como o descarte excessivo de pacotes, identificando congestionamentos.

O Traceroute funciona através da modificação dos valores do campo TTL dos pacotes que envia. São enviados sempre grupos de três datagramas UDP para o destino. O primeiro grupo contém o TTL igual a 1, fazendo com que o primeiro roteador do caminho descarte o pacote, retornando para o remetente uma mensagem de erro ICMP de tempo excedido, contendo dados sobre este roteador, como o número IP, o nome, e o RTT de cada um dos três pacotes.

Em seguida, é enviado mais um grupo de pacotes, desta vez com o campo TTL com valor igual a 2. Se atingir outro roteador, este irá retornar, de maneira semelhante ao primeiro, uma mensagem de erro ICMP de tempo excedido para cada datagrama, contendo informações úteis.

A cada grupo de pacotes o TTL é incrementado, obtendo informações de cada roteador, até que se alcance o destino, permitindo a investigação de aspectos de roteamento ocorridos durante a comunicação com um determinado destino.

O tamanho padrão dos datagramas UDP enviados pela ferramenta é de 40 *bytes*, mas pode ser modificado. A porta padrão UDP de destino utilizada é a de número 33434. Outros valores como o tempo de espera para cada pacote ou os TTLs final e inicial também podem ser configurados.

6.7.3 A ferramenta Tracepath

Bastante similar ao Traceroute, mas com algumas informações a mais. Além de exibir o TTL para cada pacote e de recolher dados acerca de cada roteador do caminho, o Tracepath mostra informações sobre o MTU (*Maximum Transfer Unit*) de cada enlace, e ao final da execução exibe um resumo contendo o número de saltos até o destino, e o tamanho do MTU primário, que é o MTU do enlace local da interface de rede da máquina de origem.

Ao invés de enviar grupos com três pacotes, o Tracepath envia apenas um pacote de cada vez. Além disso, não possui uma porta UDP padrão: se não for informada uma porta específica, a escolha será aleatória.

6.7.4 Testando a capacidade de comunicação do 6ix

Falamos bastante sobre os métodos de acesso do 6ix Linux. Para assegurar a sua real capacidade de comunicação através do protocolo IPv6, foram realizados alguns testes simples de conectividade, que ainda serviram para mostrar na prática, a configuração dos túneis IPv6.

Com acesso à Internet, e com um túnel IPv6 configurado, já é possível navegar por páginas *web* hospedadas na rede IPv6, desde que o navegador possua suporte adequado. O Mozilla Firefox, navegador de código livre, multi-plataforma e um dos mais conhecidos para o Linux, possui suporte ao IPv6 [51]. É este o navegador utilizado para acessar a *web* IPv6, no 6ix Linux.

Se o sítio estiver hospedado na rede IPv6, e uma conexão IPv6 estiver configurada no *host*, o protocolo terá preferência, e será utilizado automaticamente para navegar pelas páginas. Mas o programa de navegação não mostra se a conexão utilizada para acessar uma dessas páginas utiliza um endereço IP versão 4 ou 6. Dessa forma, é preciso se certificar de que o endereço IPv6 está realmente sendo utilizado para navegar. Existem diversas maneiras de se fazer isso.

Um jeito fácil de assegurar-se que a conexão IPv6 está sendo utilizada, é acessar uma página exclusivamente IPv6. Existem também algumas páginas que ao serem acessadas, exibem o endereço IP do visitante, como é o caso do *site www.ipv6.org*, criado com o intuito de divulgar informações sobre o IPv6. Existe ainda uma terceira maneira de verificar a utilização da conexão IPv6. No sítio do projeto Kame, *www.kame.net*, existe uma imagem de uma tartaruga, na página principal. Se o usuário utiliza uma conexão IPv4, a tartaruga é apresentada de forma estática, e uma mensagem avisa que “se você migrar para IPv6, poderá ver a tartaruga dançante” (Figura 23). Se a conexão for feita através do IPv6, será carregada a tartaruga dançante – mascote do projeto, e uma mensagem de boas-vindas.

O IPv6 pode então ser utilizado para acessar a *web*, no entanto a quantidade de páginas preparada para lidar com o protocolo ainda é ínfima, e em sua maioria são sites de projetos ou de abordagem específica sobre IPv6. Não há, então, grande atrativo na navegação *web* através do IPv6, que inclusive é até mais lenta através do protocolo, devido aos atrasos causados pelo tunelamento.

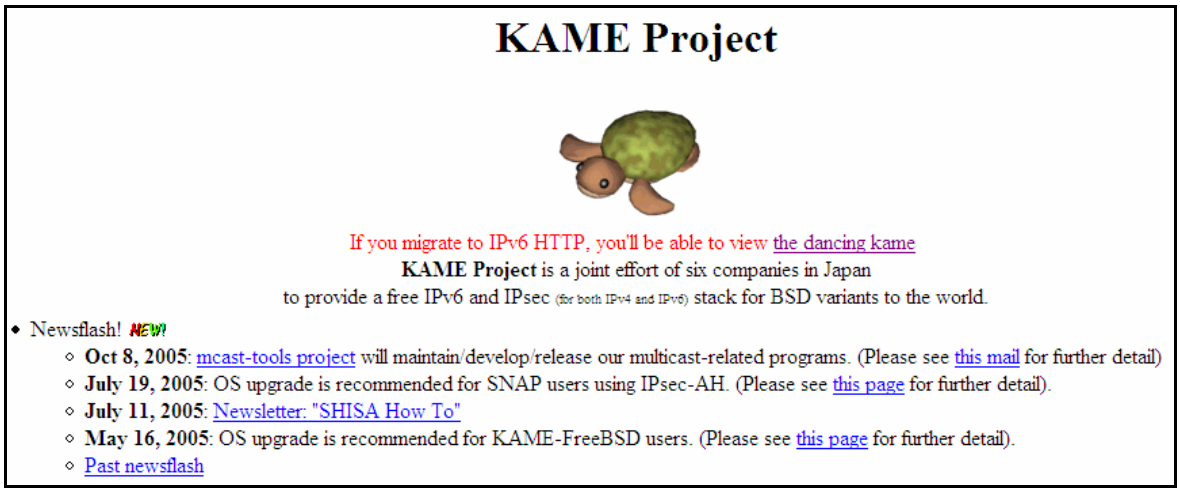

Figura 23. Página principal do sítio do projeto KAME.

Mas o IPv6 pode ser utilizado também para qualquer outro tipo de comunicação, além da navegação *web*. Basta que a aplicação suporte a nova versão do protocolo. Podemos utilizá-lo, por exemplo, para estabelecer uma conexão SSH.

Um dos nossos experimentos consistiu na utilização do 6ix Linux para estabelecimento de túneis IPv6, em duas máquinas distintas e distantes, fazendo-as se comunicarem através do protocolo e inclusive estabelecendo uma comunicação SSH. O SSH é um protocolo de rede e uma aplicação que permitem a execução de comandos em um *host* remoto, de forma semelhante ao Telnet, mas com criptografia [13]. Também executamos as ferramentas Ping, Traceroute e Tracepath. Durante os testes, os *firewalls* de ambas as máquinas foram mantidos desabilitados, para evitar possíveis conflitos.

Ambos os *hosts* possuem acesso à Internet através de conexão banda larga ADSL (*Asymmetric Digital Subscriber Lines*), de 300 *Kbps*. Esse tipo de conexão disponibiliza ao usuário um endereço IP real, embora temporário. Assim, a cada conexão, é realizada uma atribuição dinâmica de um endereço roteável para o *host*. A conexão ADSL do *hosts* utiliza o protocolo PPPoE (*point-to-point protocol over ethernet*). O linux cria uma interface de rede virtual, do tipo *ppp*, para essa espécie de conexão [23].

No experimento, havia apenas uma placa de rede em cada *host*. Assim, com a conexão à Internet estabelecida, cada uma apresentava uma interface de rede física, *eth0*, e uma interface virtual, *ppp0*. As interfaces do tipo *eth* apresentavam endereços IPv4 de rede privados, utilizados para a comunicação local com os modems ADSL. As interfaces *ppp0* continham os endereços IP reais utilizados para acesso à Internet.

Dando início à realização dos testes, escolhemos uma das máquinas, configurando-a com um túnel *6to4*, e preparando-a para agir como servidora SSH, para ser acessada remotamente. Chamaremos esta máquina, daqui por diante, de Máquina1.

Criado o túnel, através do comando ***ifconfig*** constatamos a existência de uma nova interface de rede, a *sit1*. A interface de acesso à Internet, *ppp0*, mostrava que o endereço IPv4 naquele momento era *201.8.213.6*. A interface *sit1*, por sua vez, indicava a presença de um endereço IPv6 de escopo global, *2002:c908:d506:0:0:0:0:2/128*.

Como pode ser visto na Figura 24, o endereço IPv6 de escopo global, criado com a ajuda do *script 6to4* desenvolvido em Python, utiliza o endereço IPv4 convertido em hexadecimal, para preencher os 32 *bits* restantes do prefixo do endereço. O valor 128 que aparece ao final do endereço é o prefixo, e indica que este é um endereço único final, de *host*.

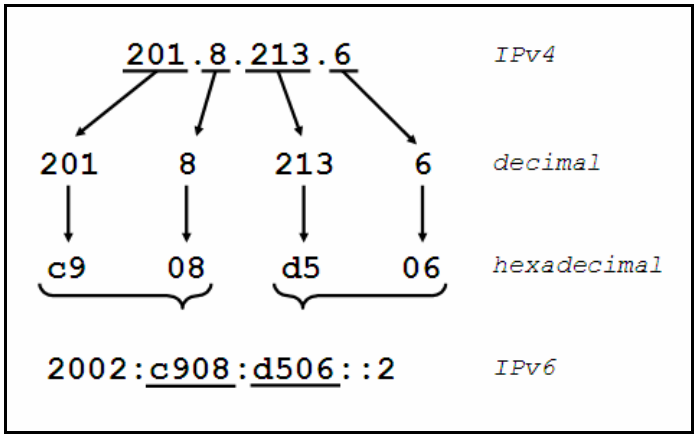

Figura 24. O endereço IPv4 é utilizado na montagem do endereço global IPv6 do tipo *6to4*.

Com a conexão IPv6 estabelecida, era preciso agora iniciar o serviço SSH. Com privilégios de administrador, no terminal, foi digitado o comando ***sshd -6*** , deixando então habilitado o servidor SSH na Máquina 1, pronta para ser acessada remotamente. A opção *-6* do comando força o servidor a aceitar apenas o protocolo IPv6 para a comunicação.

Para constatar que o serviço estava operante, utilizamos a ferramenta Nmap. Como mostra a Figura 25, com essa ferramenta foi efetuada uma varredura no próprio endereço IPv6 local, à procura das portas TCP que estavam abertas. A porta SSH padrão é a 22.

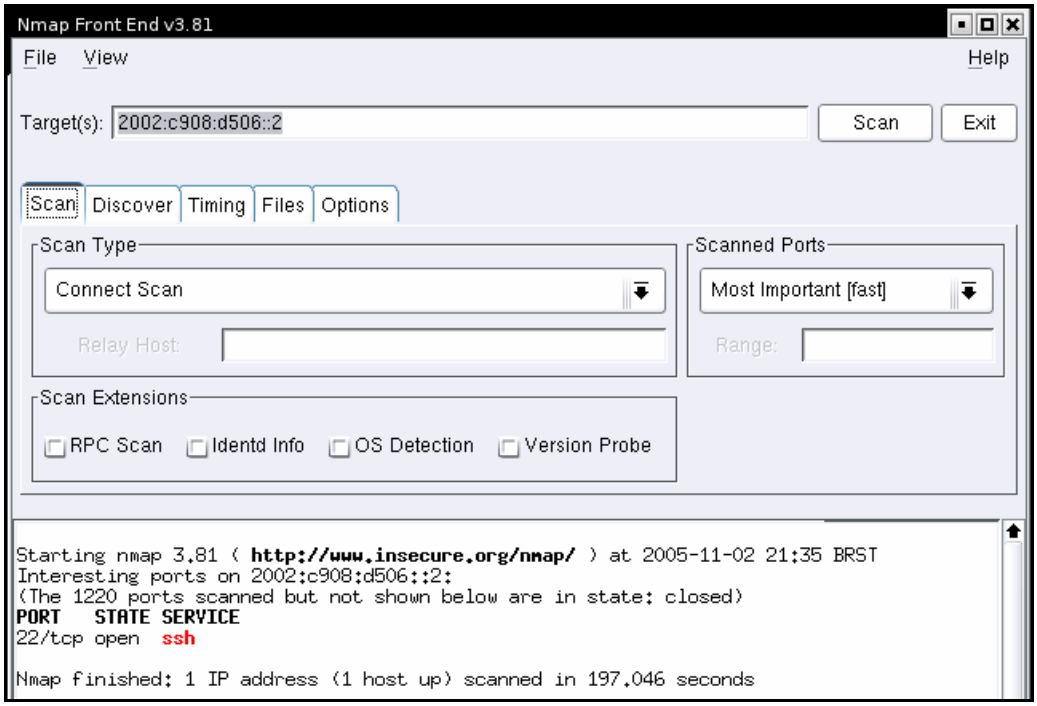

Figura 25. Tela do NmapFE, mostrando o resultado da varredura em busca da porta TCP utilizada pelo servidor SSH.

No outro *host*, o qual chamaremos de Máquina 2, agimos de forma semelhante à que foi realizada para a configuração da Máquina 1. Estabelecemos uma conexão à Internet através de ADSL, e configuramos um túnel IPv6. O endereço IPv4 real da máquina era *201.32.157.21*. O endereço IPv6-6to4 era *2002:c920:9d15:0:0:0:0:2/128*.

Para certificar-se de que seria possível a comunicação entre as duas máquinas, utilizamos a ferramenta Ping, através do comando ***ping6 –c 5 2002:c908:d506::2***, cujo resultado é mostrado no Quadro 7, indicando que a Máquina 1 estava acessível.

Quadro 7. Resultado do comando Ping executado na Máquina 2, com destino à Máquina 1.

```
PING 2002:c908:d506::2(2002:c908:d506::2) 56 data bytes
64 bytes from 2002:c908:d506::2: icmp_seq=2 ttl=64 time=74.1 ms
64 bytes from 2002:c908:d506::2: icmp_seq=3 ttl=64 time=72.2 ms
64 bytes from 2002:c908:d506::2: icmp_seq=4 ttl=64 time=75.2 ms
64 bytes from 2002:c908:d506::2: icmp_seq=5 ttl=64 time=70.5 ms

--- 2002:c908:d506::2 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4003ms
rtt min/avg/max/mdev = 70.543/73.042/75.213/1.815 ms
```

Em seguida, utilizamos novamente a ferramenta Nmap, com os mesmos parâmetros executados na Máquina 1, obtendo resultado semelhante ao da Figura 25, mostrando que a porta SSH estava aberta, e a conexão poderia ser estabelecida. Em seguida, digitamos o comando ***ssh -6 2002:c908:d506::2***. Foi pedida a senha do usuário, e em seguida já estava acessível o terminal remoto, sem nenhuma dificuldade, tal qual mostra a Figura 26.

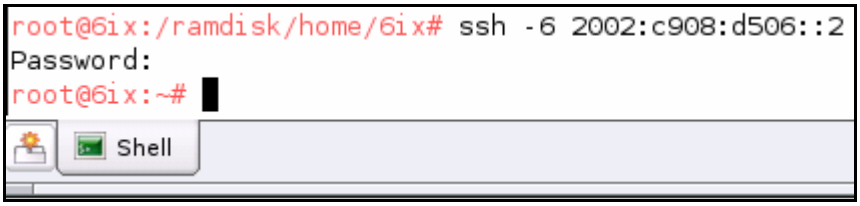

Figura 26. Pedaço da tela de um terminal na Máquina 2, mostrando o estabelecimento da conexão SSH com a Máquina 1.

Ficou provada a capacidade de comunicação, através do protocolo IPv6, do 6ix Linux. Mas os testes não acabaram. Para entender um pouco melhor o funcionamento dos túneis e clarear algumas dúvidas, utilizamos as ferramentas Traceroute e Tracepath, primeiramente com o túnel *6to4* já estabelecido, e em seguida com um túnel *freenet6*. A Máquina 1 foi mantida com o mesmo túnel *6to4*.

As duas ferramentas foram utilizadas para verificar a quantidade de saltos percorrida pelos pacotes partindo da Máquina 2 para a Máquina 1, e vice-versa. O Quadro 8 mostra as informações obtidas com a ferramenta *traceroute6*, ainda utilizando um túnel *6to4* na Máquina 2. Como o *relay router* utilizado pelas duas máquinas era o mesmo, não apareceram saltos intermediários. As duas máquinas pertenciam a uma mesma rede IPv6.

Quadro 8. Retorno da execução da ferramenta Traceroute com um túnel *6to4*.

```
traceroute to 2002:c908:d506::2 from 2002:c920:9d15::2, 30 hops max, 16 byte packets

 1  2002:c908:d506::2 (2002:c908:d506::2)      66.113 ms        66.134 ms        67.737 ms
```

Já com o túnel *freenet6*, os resultados obtidos com a ajuda das ferramentas Traceroute e Tracepath foram um pouco diferentes. Antes de examiná-los, vamos dar uma olhada nos endereços da Máquina 2. Com o túnel *freenet6* configurado, ocorre situação parecida à descrita anteriormente, para o túnel *6to4*. As interfaces de rede *eth0* e *ppp0* continuaram com os mesmos endereços. Apenas o endereço IPv6 da interface *sit1* é que mudou, para *2001:5c0:8fff:ffff:8000:0:c920:9d15/128*.

Esse endereço global foi atribuído pelo *tunnel broker*, que disponibiliza endereços com prefixo *2001:5c0:8fff:ffff::/64* para conexões anônimas, sem autenticação. Se fosse negociado um túnel com autenticação de usuário, o prefixo adotado seria *2001:5c0:8fff:fffe::/64*. Os 64 *bits* menos significativos, servem para identificar o *host*. Se a conexão é feita de maneira anônima, essa identificação do *host freenet6* segue um padrão no qual o IPv4 do *host* é utilizado para preencher os 32 *bits* mais à direita do endereço (Figura 27). Dessa maneira, mesmo não tendo sido autenticado, é possível a identificação do usuário, através do seu endereço IP versão 4, coibindo a ação de usuários maliciosos.

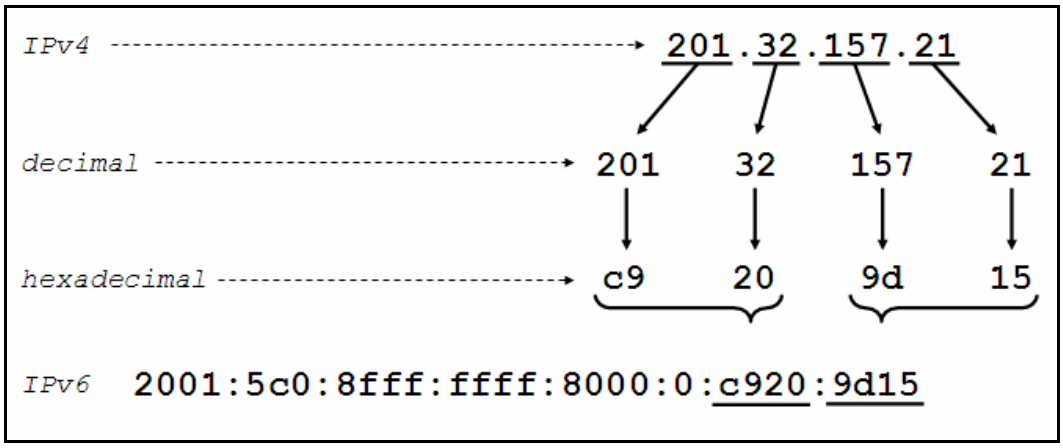

Figura 27. O endereço IPv4 é utilizado na montagem do endereço global IPv6 do tipo *freenet6*, quando o usuário é anônimo.

Analisando os dados obtidos com as ferramentas Traceroute e Tracepath, vemos que agora sim, os computadores pertencem a redes IPv6 distintas. Como mostra o Quadro 9, foram necessários cinco saltos para que os pacotes chegassem da Máquina 2 à Máquina 1.

Quadro 9. Retorno da execução da ferramenta Traceroute com um túnel *freenet6*.

```
traceroute to 2002:c908:d506::2 from 2001:5c0:8fff:ffff:8000:0:c920:9d15, 30 hops max, 16 byte packets

 1  2001:5c0:8fff:ffff:8000:1:c920:9d15                    236.761 ms  249.406 ms  243.626 ms
    (2001:5c0:8fff:ffff:8000:1:c920:9d15)
 2  2001:5c0:0:5::114 (2001:5c0:0:5::114)                  243.369 ms  235.117 ms  231.940 ms
 3  if-5-0-1.6bb1.MTT-Montreal.ipv6.teleglobe.net (2001:5a0:300::5)  226.604 ms  231.479 ms  228.605 ms
 4  gin-aeq-6bb1.ipv6.teleglobe.net (2001:5a0:600::1)      248.294 ms  251.628 ms  256.042 ms
 5  ge-0.equi6ix.asbnva01.us.bb.verio.net (2001:504:0:2::2914:1)  272.122 ms  268.471 ms  268.029 ms
 6  2002:c908:d506::2 (2002:c908:d506::2)                  559.734 ms  566.613 ms  567.870 ms
```

Cada túnel tem, portanto, uma maneira peculiar de montar seus endereços, e como veremos adiante, cada um possui suas vantagens e desvantagens em relação ao outro. Mas ambos permitem ao usuário acessar a Internet IPv6 sem grandes problemas, de maneira satisfatória.

6.7.5 Avaliando e comparando os métodos de acesso IPv6 do 6ix Linux

O intuito deste experimento foi comparar os desempenhos dos dois tipos de túneis, servindo como base para as afirmações sobre as vantagens e desvantagens de cada um em relação ao outro.

O experimento consistiu basicamente do exame dos resultados da execução dos comandos *ping6*, *traceroute6* e *tracepath6*. A metodologia adotada para a realização dos testes de Ping foi manter as opções da ferramenta – como tamanho do pacote e o tempo entre envio de pacotes, com as suas configurações originais, efetuando duas rodadas de execução para cada tipo de túnel, cada uma com 50 pacotes enviados. Apenas o valor do campo TTL inicial foi modificado, para 64 saltos.

O tamanho padrão do pacote enviado pela ferramenta é adequado por não ser grande demais – o que exigiria mais recursos dos roteadores e geraria uma maior perda de pacotes e maiores atrasos, e nem tão pequeno também, simulando com mais eficácia o comportamento da rede. A intenção de executar a ferramenta duas vezes para cada método de acesso foi a de assegurar que os resultados não eram casos isolados, decorrentes da intensidade de utilização em determinado momento da rede. Por isso a primeira execução para cada túnel foi realizada no dia 27 de outubro de 2005, por volta da 10:00 horas da manhã, e a segunda somente no dia seguinte, perto das 17:00 horas da tarde. Os valores seriam mais confiáveis ainda se o procedimento fosse repetido mais vezes, e durante um período de tempo mais extenso, como uma semana ou um mês, permitindo inclusive utilizar métodos estatísticos na avaliação dos resultados.

Para cada execução do Ping foi gerado um gráfico, para permitir uma melhor análise dos resultados. As ferramentas Traceroute e Tracepath serviram apenas como aporte para avaliação dos resultados das execuções do comando Ping. Todos os testes foram efetuados tendo um destino em comum, o sítio *www.ipv6.org*, já que este responde sem problemas a requisições ICMPv6 *echo request*.

O comando Ping utilizado foi ***ping6 -c 50 -t 64 www.ipv6.org***. O comando Traceroute foi ***traceroute6 www.ipv6.org***, e o comando Tracepath foi ***tracepath6 www.ipv6.org***.

Vamos analisar primeiro as execuções do Ping referentes ao tunelamento *6to4*. Na primeira sessão (Figura 28), a taxa de perda de pacotes foi baixa, de 6%, o que corresponde a 3 pacotes que não chegaram ao seu destino ou não obtiveram resposta em tempo hábil. O menor RTT foi de 348*ms*, o máximo foi de 355*ms*, e a média ficou em torno dos 350*ms*. Todos os pacotes mantiveram-se numa faixa bem próxima à média, sem grandes desvios, o que demonstra um comportamento uniforme no roteamento dos pacotes. O retorno do comando Ping contendo os dados da execução utilizados para a construção do gráfico pode ser conferido no Apêndice D, ao final deste trabalho.

Já na segunda sessão Ping com tunelamento *6to4* (Figura 29), a taxa de perda foi menor ainda: apenas um pacote foi perdido (2%). O valor mínimo do RTT foi de 348*ms*, o máximo disparou e foi para 412*ms*, e a média manteve-se em torno dos 352*ms*. O valor de RTT máximo foi alto em relação aos demais, provavelmente devido a algum pico momentâneo ao longo do caminho. Afinal, como já foi mencionado na Subseção 6.7.1, os pacotes ICMP que são utilizados pelo Ping não possuem nenhuma prioridade nas filas dos roteadores, sendo os últimos a serem encaminhados em momentos de tráfego mais intenso.

Mas essa discrepancia não voltou a acontecer, e os demais valores comportaram-se de maneira uniforme, mantendo-se próximos à média, que inclusive sofreu um desvio e foi elevada, devido ao valor máximo de RTT ter sido tão discordante dos demais. O comportamento do 20º pacote, portanto, praticamente não interfere nos resultados, por ter sido um caso isolado. O retorno do comando Ping contendo os dados da execução utilizados para a construção do gráfico pode ser conferido no Apêndice E, ao final deste trabalho.

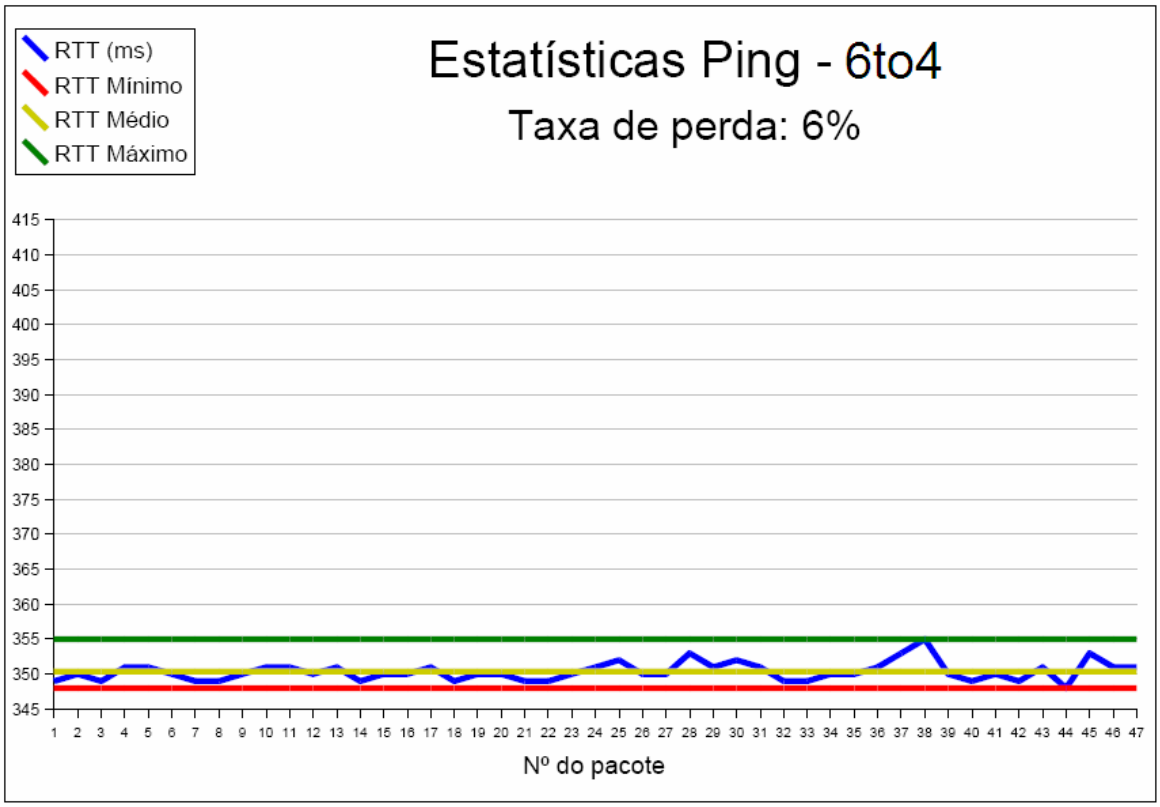

Figura 28. Gráfico mostrando o comportamento da primeira sessão Ping utilizando o tunelamento *6to4*.

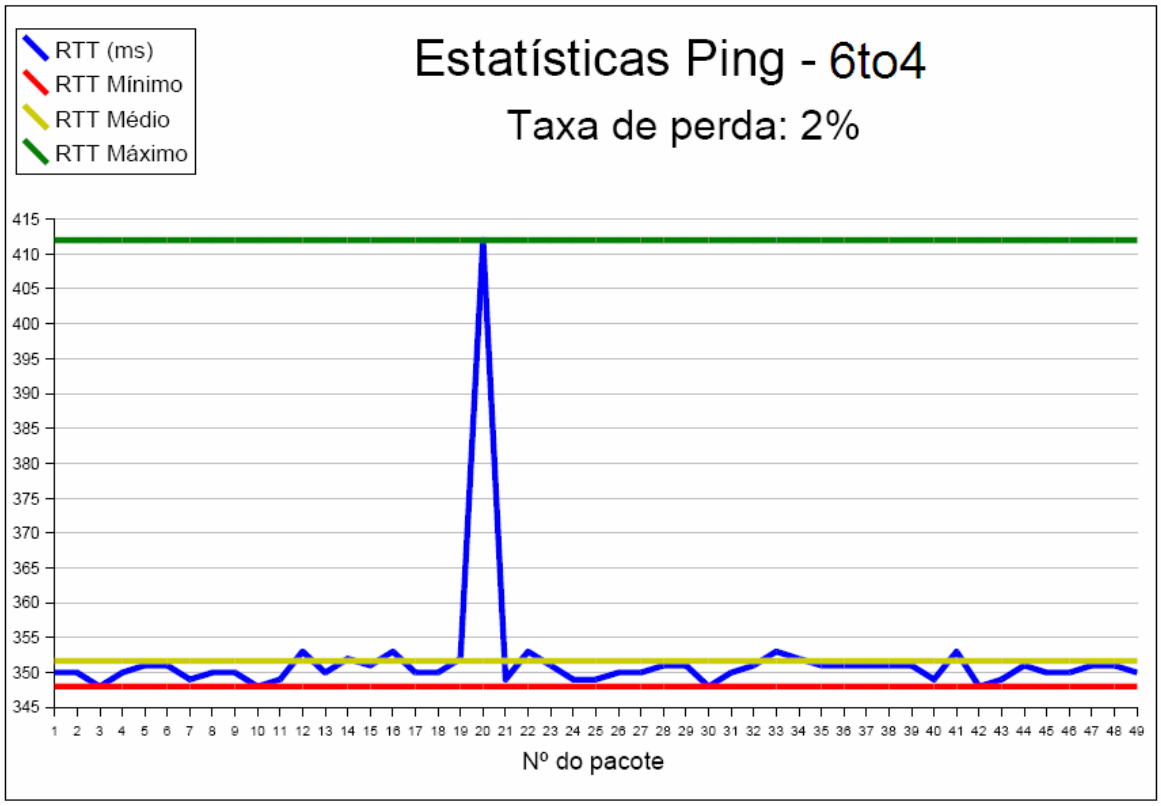

Figura 29. Gráfico mostrando o comportamento da segunda sessão Ping utilizando o tunelamento *6to4*.

Nas duas rodadas *6to4*, o valor final do TTL foi 59, indicando que para chegar de volta do seu destino, os pacotes tiveram que passar por 5 roteadores – o valor inicial do TTL era 64, logo: *64 - 59 = 5 saltos*. É preciso lembrar que só são contados os saltos IPv6, ou seja, o caminho percorrido na ida até o *relay router* pela rede IPv4 foi contabilizado apenas como um único salto, tendo sido o TTL decrementado pela primeira vez justamente ao alcançar o *relay router*. Na volta, o *relay router* foi contabilizado como o último salto. A saída do comando Traceroute, semelhante nas duas execuções, atesta as afirmações, mostrando apenas saltos com endereço IPv6, como mostra o Quadro 10.

Quadro 10. Retorno em tela de um comando *traceroute6*, executado através de um túnel *6to4*.

```
 1  swiIX1.switch.ch (2001:620:0:c000::9)                    307.596 ms  306.106 ms  308.007 ms
 2  swiEZ2-10GE-1-3.switch.ch (2001:620:0:c00a::1)           307.838 ms  308.947 ms  308.592 ms
 3  swiLS2-10GE-1-1.switch.ch (2001:620:0:c03c::1)           312.872 ms  312.857 ms      *
 4  swiCE2-10GE-1-3.switch.ch (2001:620:0:c006::1)           310.901 ms  312.842 ms  315.902 ms
 5  switch.ch1.ch.geant.net (2001:798:2012:10aa::1)          313.335 ms  311.332 ms  312.805 ms
 6  ch.fr1.fr.geant.net (2001:798:20cc:1201:1601::2)         319.628 ms  319.448 ms  341.206 ms
 7  fr.uk1.uk.geant.net (2001:798:20cc:1601:2801::2)         360.854 ms  361.982 ms  362.682 ms
 8  uk.se1.se.geant.net (2001:798:20cc:2501:2801::1)         360.812 ms  360.837 ms  360.963 ms
 9  nordunet-gw.se1.se.geant.net (2001:798:2025:10aa::2)     354.356 ms  354.239 ms  353.683 ms
10  sw-gw.nordu.net (2001:948:0:f025::1)                     540.479 ms  427.18 ms   405.631 ms
11  kthnoc7.sunet.se (2001:948:0:f03c::2)                    344.357 ms      *       348.705 ms
12  kthnoc4.sunet.se (2001:6b0:1e:1::1)                      346.026 ms  347.603 ms  347.646 ms
13  kth1-v6.sunet.se (2001:6b0:8::2839)                      346.026 ms      *       345.177 ms
14  cn5.gw.kth.se (2001:6b0:1::1:5)                          345.991 ms  346.083 ms  347.589 ms
15  igloo.stacken.kth.se (2001:6b0:1:ea:202:a5ff:fecd:13a6)  349.363 ms  347.679 ms  349.311 ms
```

Olhando a saída do comando *traceroute6*, inicialmente podemos pensar que a quantidade de saltos difere da mostrada pelo *ping6*. No entanto, através do retorno do comando *tracepath6*, mostrado no Quadro 11, vemos na última linha que a quantidade de saltos de ida é maior em relação à quantidade de saltos da volta. Isso acontece devido a uma assimetria no canal, que significa que o caminho de retorno dos pacotes não é necessariamente o mesmo da ida, podendo passar por roteadores diferentes. Como o TTL mostrado na saída do comando *ping6* é o da resposta, o importante é analisar a quantidade de saltos do caminho de volta. Através da saída do comando Tracepath, notamos também que não há mudanças no tamanho inicial do MTU, que é de 1472 *bytes*.

Quadro 11. Retorno em tela de um comando *tracepath6*, executado através de um túnel *6to4*.

```
 1?: [LOCALHOST]                                                 pmtu 1472
 1:  swiIX1.switch.ch                                  445.400ms
 2:  swiEZ2-10GE-1-3.switch.ch                         445.995ms
 3:  swiLS2-10GE-1-1.switch.ch           asymm  2      450.624ms
 4:  swiCE2-10GE-1-3.switch.ch           asymm  1      450.638ms
 5:  switch.ch1.ch.geant.net             asymm  3      450.625ms
 6:  ch.fr1.fr.geant.net                 asymm  4      460.495ms
 7:  fr.uk1.uk.geant.net                 asymm  5      498.689ms
 8:  uk.se1.se.geant.net                 asymm  4      500.375ms
 9:  nordunet-gw.se1.se.geant.net        asymm  6      492.620ms
10:  sw-gw.nordu.net                     asymm  6      489.550ms
11:  no reply
12:  kthnoc4.sunet.se                    asymm  3      534.201ms
13:  no reply
14:  cn5.gw.kth.se                       asymm  5      490.159ms
15:  igloo.stacken.kth.se                asymm  6      488.688ms      reached
     Resume: pmtu 1472 hops 15 back 6
```

Passemos agora à avaliação das rodadas Ping executadas com um túnel *freenet6* configurado. A primeira sessão (Figura 30) mostrou uma perda de pacotes de 10% (5 pacotes). O RTT mínimo foi de 735*ms*, a média foi de aproximadamente 739*ms* e o máximo chegou a 753*ms*. O RTT máximo foi um pouco dispar em relação aos demais, mas nada significante, além de ter sido um caso isolado, e portanto não interfere na avaliação, pois no geral a variação em torno da média foi bem estável e não muito grande. O retorno do comando Ping contendo os dados da execução utilizados para a construção do gráfico pode ser conferido no Apêndice F, ao final deste trabalho.

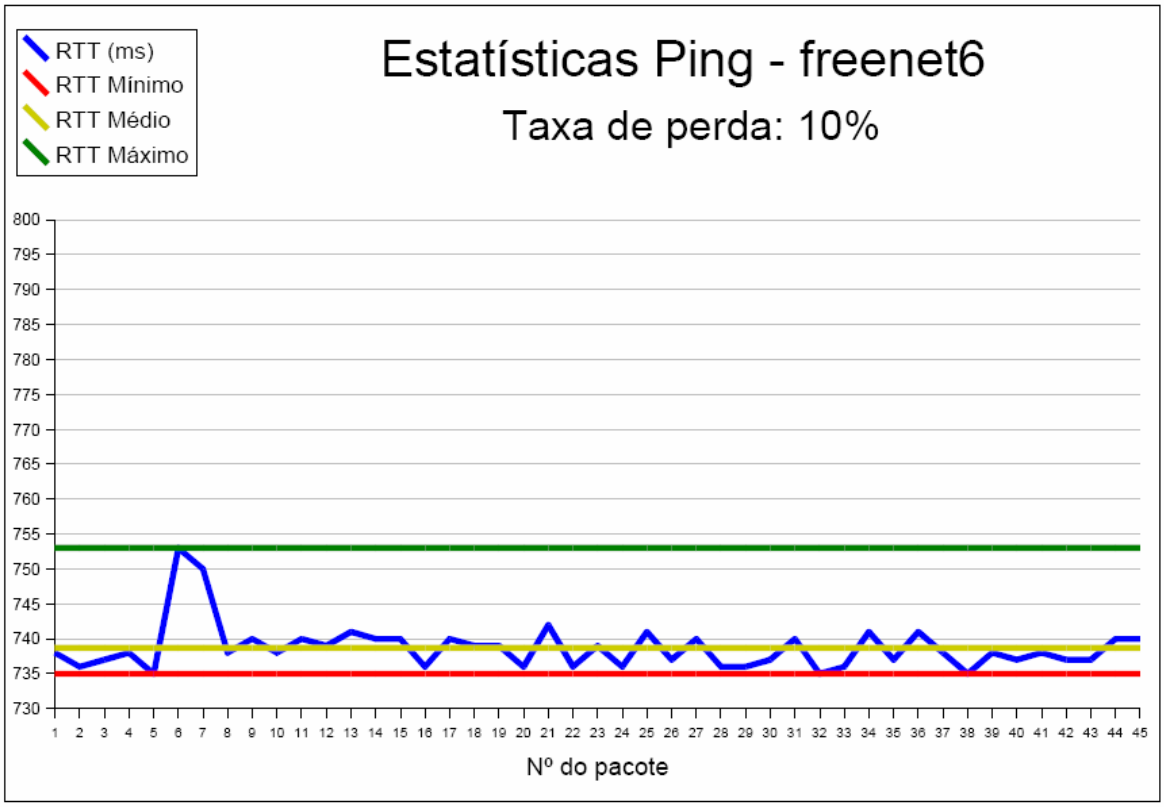

Figura 30. Gráfico mostrando o comportamento da primeira sessão Ping utilizando o tunelamento *freenet6*.

Na segunda rodada (Figura 31), executada no dia seguinte, a taxa de perda de pacotes manteve-se praticamente estável, em 12% (6 pacotes). Os valores dos tempos de ida e volta também não variaram muito. O RTT mínimo foi de 734*ms*, o máximo chegou a 755*ms*, e a média manteve-se em torno de 739*ms*. Dessa vez, notamos alguns pacotes com um RTT maior do que os demais, o que indica uma instabilidade um pouco maior na transmissão como talvez um pico de tráfego momentâneo ao longo do trajeto. Provavelmente devido à essa inconstância é que tenha sido alcançada a maior perda de pacotes dentre as 4 rodadas do teste. O retorno do comando Ping contendo os dados da execução utilizados para a construção do gráfico pode ser conferido no Apêndice G, ao final deste trabalho.

Nas duas execuções do comando Ping relativas ao túnel *freenet6*, obtivemos TTL igual a 32, indicando que para voltar de seu destino, os pacotes passaram por 32 roteadores IPv6. No entanto, como vemos no Quadro 12, a saída do comando Traceroute nos mostra a presença de apenas 21 saltos, incluindo o destino.

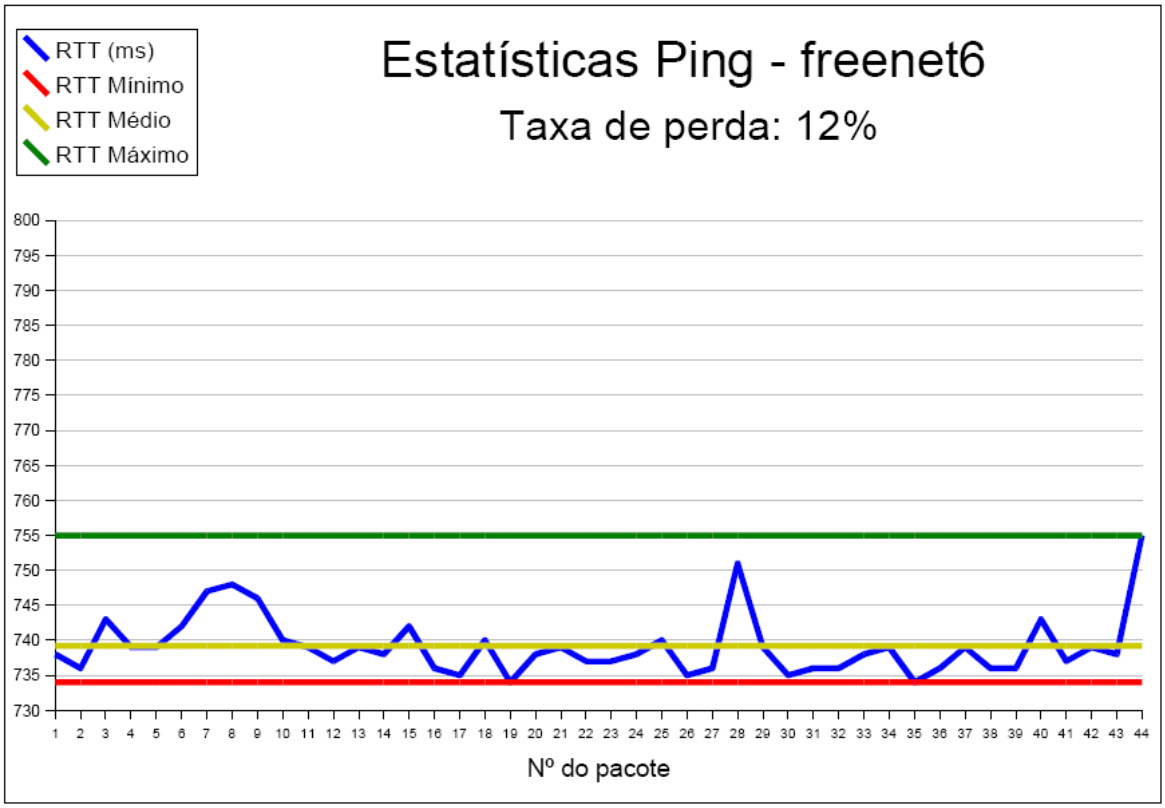

Figura 31. Gráfico mostrando o comportamento da segunda sessão Ping utilizando o tunelamento *freenet6*.

Quadro 12. Retorno em tela do comando *traceroute6*, executado através de túnel *freenet6*.

1	2001:5c0:8fff:ffff:8000:1:c8a4:ad5d (2001:5c0:8fff:ffff:8000:1:c8a4:ad5d)	208.235 ms	209.962 ms	206.708 ms
2	* 2001:5c0:0:5::114 (2001:5c0:0:5::114)	211.604 ms	*	
3	sl-bb1v6-nyc-t-23.sprintv6.net (3ffe:2900:2001:5::1)	249.561 ms	*	251.011 ms
4	sl-bb1v6-rly-t-1003.sprintv6.net (2001:440:1239:100a::2)	254.909 ms	*	254.470 ms
5	3ffe:2900:b:e::2 (3ffe:2900:b:e::2)	315.894 ms	314.576 ms	318.737 ms
6	plt001ix06.IIJ.Net (2001:240:bb62:8000::4001)	315.867 ms	314.529 ms	316.257 ms
7	* * 2001:48b0:bb00:f007::4006 (2001:48b0:bb00:f007::4006)	315.290 ms		
8	equinix6-sjc.ip.tiscali.net (2001:504:0:1::3257:1)	316.159 ms	314.461 ms	316.202 ms
9	* so-2-0-0.pao10.ip6.tiscali.net (2001:668:0:2::501)	317.116 ms	314.56 ms	
10	so-3-0-0.chi10.ip6.tiscali.net (2001:668:0:2::1f0)	319.479 ms	322.818 ms	325.048 ms
11	so-2-0-0.chi10.ip6.tiscali.net (2001:668:0:2::390)	327.284 ms	322.853 ms	*
12	so-0-0-0.lon22.ip6.tiscali.net (2001:668:0:2::4e0)	396.954 ms	398.306 ms	*
13	2001:7f8:4:1::d1c:2 (2001:7f8:4:1::d1c:2)	394.998 ms	395.678 ms	399.025 ms
14	2001:1900:5:2::2 (2001:1900:5:2::2)	806.533 ms	733.600 ms	731.900 ms
15	nordunet-gw.se1.se.geant.net (2001:798:2025:10aa::2)	736.926 ms	733.573 ms	730.259 ms
16	sw-gw.nordu.net (2001:948:0:f025::1)	735.284 ms	733.466 ms	736.665 ms
17	kthnoc7.sunet.se (2001:948:0:f03c::2)	733.754 ms	731.899 ms	730.287 ms
18	kthnoc4.sunet.se (2001:6b0:1e:1::1)	735.650 ms	729.826 ms	735.350 ms
19	kth1-v6.sunet.se (2001:6b0:8::2839)	743.496 ms	736.818 ms	733.557 ms
20	cn5.gw.kth.se (2001:6b0:1::1:5)	736.971 ms	732.073 ms	738.427 ms
21	igloo.stacken.kth.se (2001:6b0:1:ea:202:a5ff:fecd:13a6)	733.564 ms	733.110 ms	735.304 ms

Novamente, percebemos a presença de canais assimétricos, que indicam que a quantidade de saltos percorridos do remetente ao destinatário não é a mesma percorrida no sentido inverso, pelos pacotes de resposta. Como vemos com a ajuda do comando *tracepath6*, cuja saída é mostrada no Quadro 13, a quantidade de saltos no caminho da volta é de 33 incluindo o próprio destino dos pacotes enviados originalmente, correspondendo aos valores de TTL mostrados na saída do comando *ping6*. Novamente, percebemos que não há modificações no tamanho inicial do MTU, de 1280 *bytes*, durante o caminho.

Quadro 13. Retorno em tela de um comando *tracepath6*, executado através de um túnel *freenet6*.

```
 1?: [LOCALHOST]                                          pmtu 1280
 1:  2001:5c0:8fff:ffff:8000:1:c8a4:ad5d          334.558ms
 2:  2001:5c0:0:5::114                            334.805ms
 3:  sl-bb1v6-nyc-t-23.sprintv6.net    asymm  4   382.330ms
 4:  sl-bb1v6-rly-t-1003.sprintv6.net  asymm  5   450. 45ms
 5:  no reply
 6:  plt001ix06.IIJ.Net                asymm  7   449.336ms
 7:  2001:48b0:bb00:f007::4006         asymm  6   448.221ms
 8:  equinix6-sjc.ip.tiscali.net       asymm  6   445.787ms
 9:  so-2-0-0.pao10.ip6.tiscali.net    asymm  7   452.277ms
10:  so-3-0-0.chi10.ip6.tiscali.net     asymm  7   455.691ms
11:  so-0-1-0.nyc33.ip6.tiscali.net    asymm  7   452.292ms
12:  no reply
13:  2001:7f8:4:1::d1c:2               asymm  8   525.474ms
14:  2001:1900:5:2::2                  asymm 27   887.953ms
15:  nordunet-gw.se1.se.geant.net      asymm 28   886. 37ms
16:  sw-gw.nordu.net                   asymm 28   886.240ms
17:  kthnoc7.sunet.se                  asymm 29   889.530ms
18:  kthnoc4.sunet.se                  asymm 30   886.278ms
19:  kth1-v6.sunet.se                  asymm 31   890.513ms
20:  cn5.gw.kth.se                     asymm 32   889.577ms
21:  igloo.stacken.kth.se              asymm 33   889.863ms   reached
     Resume: pmtu 1280 hops 21 back 33
```

Mostrados os valores de todas as execuções – os quais estão listados de forma resumida na Tabela 2 abaixo, podemos agora realizar algumas comparações.

Tabela 2. Resumo contendo os principais valores obtidos no experimento.

			Tipo de túnel	
			6to4	freenet6
Ping	1ª Rodada	RTT Mínimo	348 *ms*	735 *ms*
		RTT Médio	350 *ms*	739 *ms*
		RTT Máximo	355 *ms*	753 *ms*
		Pacotes perdidos	6% (3 pacotes)	10% (5 pacotes)
	2ª Rodada	RTT Mínimo	348 *ms*	734 *ms*
		RTT Médio	352 *ms*	739 *ms*
		RTT Máximo	412 *ms*	755 *ms*
		Pacotes perdidos	2% (1 pacote)	12% (6 pacotes)
Número de saltos percorridos na volta			5	32
MTU Primário			1472 bytes	1280 bytes

Vimos que em nenhum caso ocorreu mudança no tamanho inicial do MTU, o chamado MTU primário. Assim, não houve gasto de tempo com o reenpacotamento e reenvio de pacotes com novos valores de MTU, o que acarretaria em perda de desempenho.

Os valores de TTL relativos ao túnel *freenet6* foram praticamente o dobro daqueles relativos ao túnel *6to4*, e a perda de pacotes também foi maior. Afirmar o motivo exato dessas diferenças é um trabalho complexo, que envolve muitas variáveis, como: atrasos em filas nos roteadores, devido a congestionamentos; velocidade de processamento dos roteadores envolvidos; interferências nos canais; dentre outras. No entanto, podemos verificar alguns agravantes.

A quantidade de saltos percorridos pelos pacotes através do túnel *freenet6* é bem maior do que a que percorreram os pacotes do túnel *6to4*. Isto com certeza gerou algum impacto nos tempos de resposta dos pacotes.

Enquanto existe apenas um único servidor *tunnel broker*, para túneis *freenet6*, a quantidade de *relay routers 6to4* espalhados geograficamente por todo o globo já passa de vinte. Com uma quantidade assim tão grande, e cada vez maior, de *relay routers*, fica mais fácil encontrar algum que esteja mais bem localizado do que *o tunnel broker freenet6*.

Neste experimento, o destino *www.ipv6.org* está localizado na Europa. Neste caso, o túnel *6to4* obteve melhor desempenho em relação ao túnel *freenet6*, cujo servidor encontra-se no Canadá. Contudo, se o destino escolhido estivesse, por exemplo, também presente no Canadá, talvez o desempenho do túnel *freenet6* pudesse ser superior.

O túnel *6to4* obteve, portanto, vantagem no quesito desempenho, em relação ao túnel *freenet6*, mostrando-se mais ágil mesmo com o acréscimo de tempo existente devido ao redirecionamento do endereço global *anycast relay router* (*192.88.99.1*) para o *relay router* mais próximo ao remetente.

Túneis *6to4* são mais simples de configurar e – em função do endereço global adotado para escolha do *relay router* mais adequado – independentes de servidor, o que lhes permite uma maior disponibilidade do acesso à rede IPv6, pois dificilmente não será possível encontrar um *relay router* funcionando, principalmente porque a cada dia aparecem mais servidores *6to4*, incentivando a migração para o novo protocolo.

No entanto, devido à sua simplicidade, túneis *6to4* disponibilizam poucas opções de configuração, quando comparados aos túneis *freenet6*. Além disso, túneis *freenet6* são mais confiáveis, pois o usuário tem certeza da idoneidade do servidor, e este por sua vez permite a autenticação criptografada do usuário, evitando o transporte de tráfego malicioso desconhecido.

Túneis *freenet6* também permitem a conexão de usuários que não possuam endereços IPv4 reais, devido à presença de NAT. Se há algum *gateway* de acesso à Internet, mesmo com IPs privados tais usuários podem negociar a configuração de um túnel com o *tunnel broker*, através do protocolo UDP. É preciso apenas, para tanto, modificar algumas das configurações padrões.

Notamos, portanto, que neste experimento o método de tunelamento automático, o *6to4*, apresentou um melhor desempenho em relação ao tunelamento explícitamente configurado *freenet6*, e que devido à quantidade muito superior de servidores espalhados ao redor do mundo, provavelmente apresentará melhores resultados na maioria dos casos. Entretanto, o segundo método também possui suas vantagens, e no geral os dois tipos de túnel se equiparam, possuindo vantagens e desvantagens em relação ao seu oponente, e mostrando-se aptos à utilização durante a fase de migração do IPv4 para o IPv6.

Capítulo 7

Conclusões e Trabalhos Futuros

A intenção principal deste projeto foi desenvolver uma distribuição linux *Debian-based Live-CD*, à qual se deu o nome de 6ix Linux, contendo várias ferramentas e pacotes para permitir a configuração do acesso à rede IPv6 já existente, além de outras funções. Espera-se, com a produção da distribuição, ajudar a divulgar e massificar a utilização do IPv6, permitindo a familiarização de usuários e a desmistificação de conceitos atualmente só entendidos por profissionais especializados. Tudo isso sem sequer modificar a configuração de seus computadores ou afetar seus sistemas operacionais instalados, já que o 6ix Linux, sendo uma distribuição *Live-CD*, poderá ser executado diretamente do CD, e ser facilmente manipulado para permitir o acesso ao *backbone* IPv6.

Após uma breve introdução, explicando os motivos da necessidade de migração entre versões do protocolo, foram apresentados, neste trabalho, conceitos relativos à arquitetura TCP/IP e ao protocolo IPv4 propriamente dito, sempre focando nas características relevantes para o entendimento das mudanças introduzidas pelo IPv6.

Em seguida, vimos algumas das principais características do novo protocolo da Internet, e as mudanças em relação à versão atual do protocolo IP. Também uma explanação sobre os métodos de migração entre os protocolos foi apresentada, com destaque para o método utilizado na distribuição desenvolvida, o tunelamento.

Ainda, alguns aspectos de segurança relacionados ao IPv6 foram citados, introduzindo a estrutura do IPSEC e buscando eliminar algumas dúvidas comuns relativas à utilização dessa estrutura de segurança do IPv6, como a substituição de alguns protocolos e a convivência com outros.

Feita a preparação teórica do leitor, partimos para a descrição do processo de desenvolvimento da distribuição, passando por todas as fases em detalhes, desde a customização do ambiente até a conclusão da versão final da distribuição, passando pelas fases de instalação e remoção de pacotes, e de desenvolvimento das ferramentas de configuração do acesso IPv6 e do *firewall*.

Por fim, foram efetuados alguns testes objetivando a demonstração da capacidade de comunicação IPv6 do 6ix Linux, e a comparação entre os dois métodos de tunelamento disponibilizados na distribuição, chegando à conclusão de que o método de tunelamento automático, o *6to4*, possui um melhor desempenho em relação ao tunelamento explícitamente configurado *freenet6*. Entretanto, o segundo método também possui suas vantagens, e no geral os dois tipos de túnel se equiparam, possuindo vantagens e desvantagens em relação ao seu oponente, e mostrando-se aptos à utilização durante a fase de migração do IPv4 para o IPv6.

7.1 Contribuições

Espera-se, com o desenvolvimento desta pesquisa, contribuir para o entendimento mais aprofundado de algumas das características e benefícios trazidos pelo IPv6, em relação ao IPv4, bem como mostrar uma das formas de se efetuar a transição entre as tecnologias, de maneira relativamente simples, que vem sendo utilizada por várias empresas, instituições e mesmo usuários domésticos, que é a utilização de túneis IPv4, com pacotes IPv6 encapsulados.

Assim como a migração entre os protocolos deverá ser efetuada de forma gradativa, também a adaptação dos usuários da Internet assim deve acontecer. Afinal, não deve demorar muito até que o IPv6 venha a se tornar o protocolo majoritariamente adotado na Internet, e o conhecimento prévio da nova tecnologia com certeza irá evitar transtornos e gastos excessivos com treinamento e contratação de mão-de-obra especializada, por exemplo, além de elucidar conceitos que dentro de alguns poucos anos estarão fortemente presentes em nosso dia-a-dia. Ainda, a popularização de ferramentas que permitam a adoção do IPv6 e das definições por ele introduzidas pode acelerar a fase de migração, diminuindo ainda mais o tempo estimado para que o IPv6 se torne, de fato, o novo padrão da Internet.

7.2 Trabalhos futuros

A distribuição 6ix Linux utiliza como padrão o gerenciador de ambiente gráfico KDE. No entanto, outros gerenciadores, como o GNOME ou o Window Maker, não foram removidos e podem ser utilizados alternativamente. Mas a customização do ambiente só foi realizada para o KDE, e seria interessante padronizar os demais ambientes, permitindo aos usuários utilizarem o gerenciador gráfico de sua preferência com a aparência do 6ix Linux.

A ferramenta de configuração do acesso IPv6 poderia ter sido desenvolvida com uma interface gráfica mais intuitiva. A atual versão pode causar alguma confusão na sua utilização, principalmente para usuários com baixo nível de conhecimento. Além disso, o *front-end* da aplicação poderia ser desenvolvido de maneira mais genérica, não sendo tão específico para interfaces KDE, permitindo sua utilização eficaz em ambientes como o GNOME ou Window Maker.

As ferramentas de configuração do acesso IPv6 poderiam, ainda, serem modificadas para permitir que a máquina seja configurada como *gateway*, distribuindo acesso IPv6 para toda uma rede privada, o que hoje só é possível através de configuração manual, sem a utilização das ferramentas.

A ferramenta de configuração do *firewall* é muito simples, e não dá praticamente nenhuma opção ao usuário, a não ser a escolha de portas a permanecerem abertas. É totalmente voltada para configuração de *firewall* pessoal, para a própria máquina apenas. Uma ferramenta mais elaborada poderia ser desenvolvida, inclusive com suporte a regras de redirecionamento, facilitando também a configuração do *firewall* para ambientes corporativos, o que hoje não acontece.

Outras formas de configurar o acesso à rede IPv6 poderiam ser disponibilizadas, aumentando o leque de possibilidades. Além disso, testes mais elaborados poderiam ser realizados para comparar os métodos de acesso, obtendo resultados mais semelhantes à realidade, o que não acontece com o Ping.

Também poderiam ser instaladas novas ferramentas com suporte a IPv6, bem como poderiam ser desenvolvidas ferramentas para facilitar a sua configuração e a das ferramentas já presentes na distribuição, como foram os casos do acesso IPv6 e do *firewall*. Além disso, não houve oportunidade de estudo e aprofundamento da maioria das ferramentas instaladas, não permitindo a otimização de execução das mesmas, e nem atestação de funcionamento totalmente adequado.

Uma modificação mais radical seria desenvolver o 6ix Linux diretamente a partir do Kernel Linux, sem se basear em outra distribuição, instalando apenas os pacotes necessários à distribuição. Tal medida reduziria a quantidade de pacotes não utilizados e espaço de armazenamento, aumentaria a segurança, que seria personalizada, e aumentaria o desempenho. No entanto, para continuar com a capacidade de execução a partir do CD, seriam necessárias várias modificações, de grande complexidade.

Enfim, há ainda um grande número de melhorias que poderiam ser desenvolvidas e implantadas no 6ix Linux, tornando-a uma distribuição mais completa, mais robusta, mais confiável, e até mais comercial, aumentando seu poder de penetração no mercado e as chances de adoção por um número significante de usuários, domésticos e profissionais, propiciando uma maior disseminação do novo protocolo da internet, o IPv6.

Apêndice A

Código-fonte do *script* de configuração do túnel *6to4*

```
#! /usr/bin/env python
# -*- coding: Latin-1 -*-
#############################################################################
# Script que configura automaticamente o acesso IPv6 do 6ix Linux #
#                                                                           #
# Autor: Pedro França Lima Neto                                             #
# Data:  02/09/2005                                                         #
#############################################################################

# Importa os módulos necessários
import os
import sys

# Declaração de variáveis
ip, ipv6, autoline = None, None, None
numParams = len(sys.argv)

#---------------------------------------------------------------------------------------
# Função "print_usage()"
#---------------------------------------------------------------------------------------
def print_usage():
  "Imprime a mensagem de modo de uso correto do Script"

  print """
6to4: parâmetros incorretos.
Modo de uso: 6to4 <interface> <auto> | <command>
  onde:
   <interface> Interface de conexão que possui um IP válido. Ex: ppp0
   <auto>      Define se a configuração passa a ser automática a  partir do
               próximo boot. Possíveis valores: True , False
   <command>   Executa um comando específico. Possíveis valores:
                 clear - Limpa as configurações do túnel 6to4
                 start - 'Liga' a interface sit1 ('abre' o túnel)
                 stop  - 'Desliga' a interface sit1 ('fecha' o túnel)"""

#---------------------------------------------------------------------------------------
# Função "validate_ip()"
#---------------------------------------------------------------------------------------
def validate_ip(num):
  "Verifica se o IP é válido ou não."
```

```
  parts = num.split('.')
  # Reservados CLASSE A
  if parts[0] == '10' :
    return False
  # Reservados CLASSE B
  elif parts[0] == '172' and int(parts[1]) in range(16,32) :
    return False
  # Reservados CLASSE C
  elif parts[0] == '192' and parts[1] == '168' :
    return False
  # CLASSE D e CLASSE E
  elif int(parts[0]) in range(224,255) :
    return False
  # LOOPBACK
  elif parts[0] == '127' :
    return False
  # 6to4 Relay Router anycast prefix
  elif parts [0] == '192' and parts[1] == '88' and parts[2] == '99' :
    return False
  else:
    return True

#-------------------------------------------------------------------------------
# Função "calc_ipv4()"
#-------------------------------------------------------------------------------
def calc_ipv4(interface):
  "Verifica se a interface existe. Se sim, tenta obter o endereço IPv4 dessa
interface."

  global ip
  print '\nTentando obter endereço IPv4 válido...'
  try:
    os.system('ifconfig ' + interface + ' > /tmp/temp')
  except:
    # Imprime, se a interface não existe
    print 'ERRO: Interface não existe no sistema...'
    print '\nFeche esta janela, escolha uma interface válida e tente novamente.'
    return False
  else:
    tempFile = open('/tmp/temp','r')
    lines = tempFile.readlines()
    tempFile.seek(0)
    for i in range(len(lines)):
      currentLine = tempFile.readline()
      words = currentLine.split(' ')
      for j in range(len(words)):
        # Como o IP deve ser válido, não se pode utilizar endereços reservados ou
privados
        if words[j]=='inet' and validate_ip(words[j+2]) :
          ip = words[j+2]
          print 'OK. Seu endereço IP versão 4 é: ' + ip
          tempFile.close()
          os.remove('/tmp/temp')
          return True
    else:
      tempFile.close()
      os.remove('/tmp/temp')
      # Imprime, se não for encontrado um IP válido
      print '\nERRO: Não foi encontrado um endereço IP público válido para esta
interface...'
      print '\nEscolha outra interface e tente novamente.'
      return False
```

```
#---------------------------------------------------------------------------------------
# Função "calc_ipv6()"
#---------------------------------------------------------------------------------------
def calc_ipv6():
  "Se o IPv4 válido é encontrado, calcula-se o endereço IPv6 correspondente."

  global ipv6
  print '\nCalculando endereço IPv6...'
  os.system('ipv6.sh ' + ip + '> /tmp/temp')
  tempFile = open('/tmp/temp','r')
  currentLine = tempFile.readline()
  ipv6 = currentLine[:-1] + '2'
  tempFile.close()
  os.remove('/tmp/temp')
  print 'OK. Seu endereço IP versão 6 é: ' + ipv6

#---------------------------------------------------------------------------------------
# Função "conf_file()"
#---------------------------------------------------------------------------------------
def conf_file():
  "Calculado o IPv6, é preciso adicionar as configurações do túnel ao arquivo
'/etc/network/interfaces'."

  msg="\n\n" + autoLine + "iface sit1 inet6 v4tunnel\n       address " + ipv6 + "\n
netmask 64\n     endpoint any\n     local " + ip + "\n     up ip -6 route add 2000::/3
via ::192.88.99.1 dev sit1\n     down ip -6 route flush dev sit1\n     ttl 64\n"
  print '\nAdicionando as linhas necessárias no arquivo "/etc/networks/interfaces" ...'
  print ' ---> Por segurança, é preciso "desligar" a interface sit1...'
  try:
    os.system('ifdown sit1 2> /dev/null')
  except:
    print ' ---> ERRO: Não foi possível executar o comando "ifdown sit1".'
  else:
    print ' ---> OK. Interface "desligada" com sucesso.'
  file = None
  file = open('/etc/network/interfaces','r+')
  lines = file.readlines()
  file.seek(0)
  file.truncate()
   # Se já existe um túnel, ele será substituído. As linhas são apagadas para serem
então reescritas
  for h in range(len(lines)):
    if 'iface sit1 inet6 v4tunnel' in lines[h] :
      print ' ---> Atenção: Já existe um túnel! Ele será substituído...'
      vi = h+7
      vf = h-3
      if 'auto sit1' in lines[h-1] :
        vf = h-4
       for k in range(vi,vf,-1):
        lines.pop(k)
      break
  # Acrescentam-se as novas linhas ao arquivo
  for l in range(len(lines)):
    file.write(lines[l])
  file.write(msg)
  file.close()
  print 'OK. Arquivo "/etc/networks/interfaces" editado com sucesso.'

#---------------------------------------------------------------------------------------
# Função "init_tunnel()"
#---------------------------------------------------------------------------------------
def init_tunnel():
  "Levanta-se a interface para que o túnel possa funcionar."

  print '\nTentando levantar a interface do túnel (sit1)...'
```

```
  try:
    os.system('ifup sit1 2> /dev/null')
  except:
      print ' ---> ERRO: Não foi possível executar o comando "ifup sit1".'
  print 'OK. Túnel ISATAP configurado! Bem-vindo à rede IPv6!'

#-------------------------------------------------------------------------------
#Início do Corpo principal do Script
#-------------------------------------------------------------------------------
if numParams == 3 :
  # A interface usada para pegar o endereço IPv4 válido e calcular o endereço IPv6
  interface = sys.argv[1]
  print 'Interface selecionada: ' + interface

  if sys.argv[2] == 'True' :
    #  Linha  que  automatiza  a  configuração  do  túnel  IPv6  durante  as  próximas
inicializações do computador
    autoLine = 'auto sit1\n'
    print 'Automatizar? Sim'

  elif sys.argv[2] == 'False' :
    autoLine = ''
    print 'Automatizar? Não'

  else:
    print_usage()
    sys.exit()

  if calc_ipv4(interface) :
    print  '\nATENÇÃO:  Serão  apagadas  quaisquer  configurações  efetuadas  pelo  Script
freenet6'

    print '-------------------------------------------------------------------------'
    os.system('freenet6 remove')
    print '-------------------------------------------------------------------------'
    calc_ipv6()
    conf_file()
    init_tunnel()

elif numParams == 2 :
  command = sys.argv[1]

  # Apaga as alterações efetuadas no arquivo "/etc/network/interfaces"

  if command == 'clear' :
    print 'Desfazendo alteraçoes...'
    print 'efetuando "shutdown" da interface sit1...'

    try:
      os.system('ifdown sit1 2> /dev/null')
    except:
      print ' ---> ERRO: Não foi possível executar o comando "ifdown sit1".'
    else:
      print ' ---> OK. "shutdown" efetuado com sucesso.'

    file = open('/etc/network/interfaces','r+')
    lines = file.readlines()
    file.seek(0)
    file.truncate()

    for h in range(len(lines)):
      if 'iface sit1 inet6 v4tunnel' in lines[h]:
        vi = h+7
        vf = h-3
        if 'auto sit1' in lines[h-1]:
          vf = h-4
```

```
        for k in range(vi,vf,-1):
          lines.pop(k)
        break
    for l in range(len(lines)):
      file.write(lines[l])
    file.close()
    print 'OK. Todas as configurações do túnel IPv6 (6to4) desfeitas com sucesso.'
  # 'Liga' a interface sit1

  elif command == 'start' :
    print '\nTentando levantar a interface do túnel (sit1)...'

    try:
      os.system('ifup sit1 2> /dev/null')
    except:
        print ' ---> ERRO: Não foi possível executar o comando "ifup sit1".'
    print '\nOK. Túnel ISATAP configurado! Bem-vindo à rede IPv6!'
  # 'Desliga' a interface sit1

  elif command == 'stop' :
    print 'efetuando "shutdown" da interface sit1...'

    try:
      os.system('ifdown sit1 2> /dev/null')
    except:
      print ' ---> ERRO: Não foi possível executar o comando "ifdown sit1".'
    else:
      print '\nOK. "shutdown" efetuado com sucesso.'

  else:
    print_usage()

else:
  print_usage()
```

Apêndice B

Código-fonte do *script* de configuração do túnel *freenet6*

```
#! /usr/bin/env python
# -*- coding: Latin-1 -*-
#############################################################################
# Script que configura o acesso IPv6, via freenet6,  do 6ix Linux #
#                                                                           #
# Autor: Pedro França Lima Neto                                             #
# Data:  07/09/2005                                                         #
#############################################################################

# Importa os módulos necessários
import os
import sys

# Declaração de variáveis
command = None
numParams = len(sys.argv)

#-----------------------------------------------------------------------------------------
# Função print_usage()
#-----------------------------------------------------------------------------------------
def print_usage():
  "Imprime a mensagem de modo de uso correto do Script"

  print """
freenet6: parâmetros incorretos.
Modo de uso: freenet6 <command> | start [<verbose>] [-f conf_file] [-r secs]
  onde command pode ser:
   include   Adiciona os links de inicialização para '/etc/init.d/tspc'
   remove    Remove  os  links de inicialização para '/etc/init.d/tspc'
   stop      Para  a execução do TSP Client (o túnel IPv6-freenet6 é 'fechado')
   start     Inicia a execução do TSP Client (o túnel IPv6-freenet6 é 'aberto')
  <verbose>  Nível de detalhe de  mensagens  exibidas  durante  o  processo  de
             inicialização do TSC Client. Possíveis valores: -v , -vv , -vvv
  -f         Lê o  arquivo  de  configuração  especificado,  ao invés do padrão
             (/etc/tsp/tspc.conf)
  -r         Tempo, em segundos, entre tentativas de conexão, até obter sucesso

Para outras opções referentes ao TSPC, execute "tspc -h" em um prompt shell."""
```

```
#--------------------------------------------------------------------------------
# Função start()
#--------------------------------------------------------------------------------
def start(argsList):
  "Inicia a execução do TSP Client"

  args = ''
  print '\nATENÇÃO: Serão apagadas quaisquer configurações efetuadas pelo Script 6to4'
  print '--------------------------------------------------------------------------'
  os.system('6to4 clear')
  print '--------------------------------------------------------------------------'
  for i in range(len(argsList)):
    if i != 0 :
      args += ' ' + argsList[i]
  print '\nIniciando o serviço...'
  if os.access('/etc/init.d/tspc',0) :
    os.system('tspc' + args)
    print '\nOK.'
  else:
    print ' ---> ERRO: Não foi possível executar o comando. Arquivo "/etc/init.d/tspc"
não encontrado.'

#--------------------------------------------------------------------------------
# Função stop()
#--------------------------------------------------------------------------------
def stop():
  "Para a execução do túnel IPv6 - freenet6"

  print 'Parando o serviço...'
  if os.access('/etc/init.d/tspc',0) :
    os.system('invoke-rc.d tspc stop > /dev/null')
    print '\nOK.'
  else:
    print ' ---> ERRO: Não foi possível executar o comando. Arquivo "/etc/init.d/tspc"
não encontrado.'

#--------------------------------------------------------------------------------
# Função include()
#--------------------------------------------------------------------------------
def include():
  "Adiciona os links de inicialização para '/etc/init.d/tspc'"

  print '\nATENÇÃO: Serão apagadas quaisquer configurações efetuadas pelo Script 6to4'
  print '--------------------------------------------------------------------------'
  os.system('6to4 clear')
  print '--------------------------------------------------------------------------'
  print '\nProcurando o arquivo "/etc/init.d/tspc" ...'
  if os.access('/etc/init.d/tspc',0) :
    print 'OK.'
    if not os.access('/etc/rcS.d/S41tspc',0) :
      os.system('update-rc.d tspc start 41 S . stop 35 0 6 . > /dev/null')
      print """
Adicionando os seguintes links de inicialização para "/etc/init.d/tspc" :
  /etc/rc0.d/K35tspc -> ../init.d/tspc
  /etc/rc6.d/K35tspc -> ../init.d/tspc
  /etc/rcS.d/S41tspc -> ../init.d/tspc
"""
      print 'OK'
      print 'Iniciando o serviço...'
      os.system('invoke-rc.d tspc start > /dev/null')
      print 'OK'
      print 'Os links de inicialização para "/etc/init.d/tspc" foram criados com
sucesso.'
    else:
      print ' ---> ERRO: Os links de inicialização para "/etc/init.d/tspc" já estão
configurados.'
```

```
  else:
    print ' ---> ERRO: Não foi encontrado o arquivo "/etc/init.d/tspc".'

#--------------------------------------------------------------------------------
# Função remove()
#--------------------------------------------------------------------------------
def remove():
  "Remove os links de inicialização para '/etc/init.d/tspc'"

  print 'Procurando o arquivo "/etc/init.d/tspc" ...'
  if os.access('/etc/init.d/tspc',0) :
    print 'OK.'
    print 'Parando o serviço...'
    os.system('invoke-rc.d tspc stop > /dev/null')
    print 'OK.'
    if os.access('/etc/rcS.d/S41tspc',0) :
      os.system('update-rc.d -f tspc remove > /dev/null')
      print """
Removendo os seguintes links de inicialização para "/etc/init.d/tspc" :
   /etc/rc0.d/K35tspc
   /etc/rc6.d/K35tspc
   /etc/rcS.d/S41tspc
"""
      print 'OK'
      print 'Os links de inicialização para "/etc/init.d/tspc" foram removidos com
sucesso.'
    else:
      print ' ---> ERRO: não existem links de inicialização para "/etc/init.d/tspc".'
  else:
    print ' ---> ERRO: Não foi encontrado o arquivo "/etc/init.d/tspc" ...'

#--------------------------------------------------------------------------------
#Início do Corpo principal do Script
#--------------------------------------------------------------------------------
if numParams == 2 :
  command = sys.argv[1]
  if command == 'include' :
    include()
  elif command == 'remove' :
    remove()
  elif command == 'start' :
    start(sys.argv)
  elif command == 'stop' :
    stop()
  else:
    print_usage()
elif numParams > 2 :
  command = sys.argv[1]
  if command == 'start' :
    start(sys.argv)
  else:
    print_usage()
else:
  print_usage()
```

Apêndice C

Código-fonte do *script* de configuração do *firewall* IPv6

```
#! /usr/bin/env python
# -*- coding: Latin-1 -*-
#############################################################################
# Script que configura firewall IPv6 do 6ix Linux                           #
#                                                                           #
# Autor: Pedro França Lima Neto                                             #
# Data:  27/09/2005                                                         #
#############################################################################

# Importa os módulos necessários
import os
import sys

# Declaração de variáveis
numParams = len(sys.argv)
firewallIntro = """#!/bin/bash
#############################################################
#                  6ix Firewall - IPv6                      #
#          Arquivo de configuração de regras de             #
#            firewall específicas para IPv6                 #
#############################################################
# Por padrão, para evitar quaisquer problemas de conexão,   #
# este arquivo inicialmente não contém nenhuma regra. Se    #
# você tem  alguma  experiência na manipulação de regras    #
# do   firewall  linux,  o  iptables,  edite  o  arquivo    #
# manualmente, para que se adeque às suas necessidades.     #
# Lembre-se  apenas que  o módulo do firewall específico    #
# para  IPv6  é  o  ip6tables,  de  sintaxe  similar  ao    #
# iptables. Em caso de dúvidas, consulte o manual.          #
# Para usuários iniciantes, recomenda-se a utilização da    #
# ferramenta de configuração  do  6ix Linux,  acessível     #
# através do menu IPv6 > Configurar Firewall.               #
#############################################################
# ATENÇÃO: Mesmo  modificando  o  arquivo,  deverá  ser     #
#          executado  manualmente,  cada  vez  que  a       #
#          máquina for ligada. Basta digitar,  na  linha    #
#          de comando de um terminal, o comando:            #
#          "sudo /etc/./6ixfirewall.sh".                    #
#############################################################
"""
```

```
firewallBasicRules = """
# Limpa todas as regras do firewall
ip6tables -F
# Bloqueia conexões TCP externas
ip6tables -I INPUT -i sit+ -p tcp --syn -j DROP
ip6tables -I FORWARD -i sit+ -p tcp --syn -j DROP
# Bloqueia conexões UDP externas
ip6tables -I INPUT -i sit+ -p udp -j DROP
ip6tables -I FORWARD -i sit+ -p udp -j DROP
"""

#--------------------------------------------------------------------------------
# Função "print_usage()"
#--------------------------------------------------------------------------------
def print_usage():
  "Imprime a mensagem de modo de uso correto do Script"

  print """
6ixfirewall: parâmetros incorretos.
Modo de uso: 6ixfirewall [-t <tcp>] [-u <udp>] | <clear> | <stop> | <start>
  onde:
   <tcp>    Lista de portas TCP de entrada que deverão ser abertas
            (separadas por vírgulas)
   <udp>    Lista de portas UDP de entrada que deverão ser abertas
            (separadas por vírgulas)
   <clear>  Apaga as alterações feitas no arquivo '/etc/6ixfirewall.sh'
   <stop>   Desativa o firewall
   <start>  Ativa o firewall"""

#--------------------------------------------------------------------------------
# Função "configure_rules()"
#--------------------------------------------------------------------------------
def configure_rules(tcp=None,udp=None):
  "Configura as regras do firewall."

  try:
    os.access('/home/root/6ixfirewall.sh',0)
  except:
    print '  --->  Arquivo não encontrado. Recriando...'
    os.system('cat << EOF >> /home/root/6ixfirewall.sh')
    os.system('chmod +x /home/root/6ixfirewall.sh')
  file = open('/home/root/6ixfirewall.sh','r+')
  file.write(firewallIntro)
  file.write(firewallBasicRules)
  # Lista de portas TCP a serem mantidas abertas
  if tcp != None :
    for i in range(len(tcp)):
      file.write('# Abre a porta TCP #%s\n' %tcp[i])
      file.write('ip6tables -I INPUT -i sit+ -p tcp --dport %s -j ACCEPT\n' %tcp[i])
      file.write('ip6tables -I FORWARD -i sit+ -p tcp --dport %s -j ACCEPT\n' %tcp[i])
  # Lista de portas UDP a serem mantidas abertas
  if udp != None :
    for i in range(len(udp)):
      file.write('# Abre a porta UDP #%s\n' %udp[i])
      file.write('ip6tables -I INPUT -i sit+ -p udp --dport %s -j ACCEPT\n' %udp[i])
      file.write('ip6tables -I FORWARD -i sit+ -p udp --dport %s -j ACCEPT\n' %udp[i])
  file.truncate()
  file.close()
  print "OK. Arquivo '/home/root/6ixfirewall.sh' configurado com sucesso"

#--------------------------------------------------------------------------------
# Função "clear_rules()"
#--------------------------------------------------------------------------------
def clear_rules():
  "Apaga as regras do firewall do arquivo '6ixfirewall.sh'."
```

```
  try:
    os.access('/home/root/6ixfirewall.sh',0)
  except:
    print '  --->  Arquivo não encontrado. Recriando...'
    os.system('cat << EOF >> /home/root/6ixfirewall.sh')
    os.system('chmod +x /home/root/6ixfirewall.sh')
  file = open('/home/root/6ixfirewall.sh','r+')
  file.write(firewallIntro)
  file.write('\n# Lista as regras atuais do firewall\nip6tables -L\n')
  file.truncate()
  file.close()
  print "OK. Arquivo '/home/root/6ixfirewall.sh' reescrito com sucesso."

#--------------------------------------------------------------------------------
# Função "stop_firewall()"
#--------------------------------------------------------------------------------
def stop_firewall():
  "Para a atuação do firewall."
  os.system('ip6tables -F')
  print 'OK. Firewall desativado com sucesso.'

#--------------------------------------------------------------------------------
# Função "start_firewall()"
#--------------------------------------------------------------------------------
def start_firewall():
  "Ativa o firewall."
  try:
    os.access('/home/root/6ixfirewall.sh',0)
  except:
    print "  --->  ERRO: Arquivo '6ixfirewall.sh' não encontrado."
  else:
    os.system('/home/root/./6ixfirewall.sh')
    print 'OK. Firewall Ativado com sucesso.'

#--------------------------------------------------------------------------------
#Início do Corpo principal do Script
#--------------------------------------------------------------------------------
if numParams == 5 :
  if sys.argv[1] == '-t' and sys.argv[3] == '-u':
    TCP = sys.argv[2].split(',')
    UDP = sys.argv[4].split(',')
    configure_rules(TCP,UDP)
  else:
    print_usage()
elif numParams == 3 :
  if sys.argv[1] == '-t' :
    TCP = sys.argv[2].split(',')
    configure_rules(TCP)
  elif sys.argv[1] == '-u':
    UDP = sys.argv[2].split(',')
    configure_rules(UDP)
  else:
    print_usage()
elif numParams == 2 :
  if sys.argv[1] == 'clear' :
    clear_rules()
  elif sys.argv[1] == 'stop' :
    stop_firewall()
  elif sys.argv[1] == 'start' :
    start_firewall()
  else:
    print_usage()
elif numParams == 1 :
  configure_rules()
else:
  print_usage()
```

Apêndice D

Retorno da 1ª rodada Ping (6to4)

```
PING www.ipv6.org(igloo.stacken.kth.se) 56 data bytes
64 bytes from igloo.stacken.kth.se: icmp_seq=1 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=2 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=3 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=4 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=6 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=7 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=8 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=9 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=10 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=11 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=12 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=13 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=14 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=15 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=16 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=18 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=19 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=20 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=21 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=22 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=23 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=24 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=25 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=26 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=27 ttl=59 time=352 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=28 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=29 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=30 ttl=59 time=353 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=31 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=32 ttl=59 time=352 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=34 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=35 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=36 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=37 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=38 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=39 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=40 ttl=59 time=353 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=41 ttl=59 time=355 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=42 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=43 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=44 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=45 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=46 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=47 ttl=59 time=348 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=48 ttl=59 time=353 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=49 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=50 ttl=59 time=351 ms

--- www.ipv6.org ping statistics ---
50 packets transmitted, 47 received, 6% packet loss, time 62277ms
rtt min/avg/max/mdev = 348.778/350.928/355.344/1.428 ms
```

Apêndice E

Retorno da 2ª rodada Ping (6to4)

```
PING www.ipv6.org(igloo.stacken.kth.se) 56 data bytes
64 bytes from igloo.stacken.kth.se: icmp_seq=1 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=2 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=3 ttl=59 time=348 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=4 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=5 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=6 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=7 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=8 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=9 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=10 ttl=59 time=348 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=11 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=12 ttl=59 time=353 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=13 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=14 ttl=59 time=352 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=15 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=16 ttl=59 time=353 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=17 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=18 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=19 ttl=59 time=352 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=20 ttl=59 time=412 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=21 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=22 ttl=59 time=353 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=23 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=24 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=25 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=26 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=27 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=28 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=29 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=31 ttl=59 time=348 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=32 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=33 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=34 ttl=59 time=353 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=35 ttl=59 time=352 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=36 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=37 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=38 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=39 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=40 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=41 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=42 ttl=59 time=353 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=43 ttl=59 time=348 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=44 ttl=59 time=349 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=45 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=46 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=47 ttl=59 time=350 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=48 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=49 ttl=59 time=351 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=50 ttl=59 time=350 ms

--- www.ipv6.org ping statistics ---
50 packets transmitted, 49 received, 2% packet loss, time 58734ms
rtt min/avg/max/mdev = 348.287/352.162/412.555/8.820 ms
```

Apêndice F

Retorno da 1º rodada Ping (freenet6)

PING www.ipv6.org(igloo.stacken.kth.se) 56 data bytes
64 bytes from igloo.stacken.kth.se: icmp_seq=1 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=3 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=4 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=5 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=7 ttl=32 time=735 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=8 ttl=32 time=753 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=9 ttl=32 time=750 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=10 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=11 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=12 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=13 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=14 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=15 ttl=32 time=741 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=16 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=17 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=18 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=19 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=20 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=21 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=24 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=25 ttl=32 time=742 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=26 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=27 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=28 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=29 ttl=32 time=741 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=30 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=31 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=32 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=33 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=34 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=35 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=36 ttl=32 time=735 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=37 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=38 ttl=32 time=741 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=39 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=40 ttl=32 time=741 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=41 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=42 ttl=32 time=735 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=43 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=45 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=46 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=47 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=48 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=49 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=50 ttl=32 time=740 ms

--- www.ipv6.org ping statistics ---
50 packets transmitted, 45 received, 10% packet loss, time 49145ms
rtt min/avg/max/mdev = 735.373/739.212/753.081/3.462 ms

Apêndice G

Retorno da 2º rodada Ping (freenet6)

```
PING www.ipv6.org(igloo.stacken.kth.se) 56 data bytes
64 bytes from igloo.stacken.kth.se: icmp_seq=1 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=2 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=3 ttl=32 time=743 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=4 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=6 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=7 ttl=32 time=742 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=8 ttl=32 time=747 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=9 ttl=32 time=748 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=10 ttl=32 time=746 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=11 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=12 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=13 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=14 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=15 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=16 ttl=32 time=742 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=17 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=18 ttl=32 time=735 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=19 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=20 ttl=32 time=734 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=21 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=22 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=24 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=25 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=26 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=27 ttl=32 time=740 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=28 ttl=32 time=735 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=29 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=30 ttl=32 time=751 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=31 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=32 ttl=32 time=735 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=33 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=34 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=35 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=36 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=37 ttl=32 time=734 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=39 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=42 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=43 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=44 ttl=32 time=736 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=45 ttl=32 time=743 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=46 ttl=32 time=737 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=47 ttl=32 time=739 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=49 ttl=32 time=738 ms
64 bytes from igloo.stacken.kth.se: icmp_seq=50 ttl=32 time=755 ms

--- www.ipv6.org ping statistics ---
50 packets transmitted, 44 received, 12% packet loss, time 58837ms
rtt min/avg/max/mdev = 734.889/739.732/755.866/4.394 ms
```

Bibliografia

[1] BAPTISTA, Miguel; DOMINGUES, Mónica. *DNS em IPv6*. **FCCN**, 2004. Disponível em: <http://www.fccn.pt/>

[2] BIERINGER, Peter. **Linux Ipv6 Howto**. Disponível em: <http://www.bieringer.de/linux/ipv6/index.html>

[3] BORGES, André. *Superlotação na internet?* **Computerworld**, ed. 410, 09 jun. 2004. Disponível em: <http://computerworld.uol.com.br/>

[4] CAVALHEIRO, Toni. IPv6 na Pática. **Revista Www.com.br**. n. 51, p. 30-34, set. 2004. Mensal. ISSN 1518-1561.

[5] CHILDRESS, Brian; CATHEY, Bryan; DIXON, Sara. *The Adoption of IPv6*. Artigo. **Journal of Computing Sciences in Colleges**, v. 18, ed. 4, p. 153-158, abr. 2003.

[6] CHO, Kenjiro; LUCKIE, Matthew; HUFFAKER, Bradley. *Identifying IPv6 Network Problems in the DualStack World*. **Applications, Technologies, Architectures, and Protocols for Computer Communication**. *In:* **Proceedings of the ACM SIGCOMM workshop on Network troubleshooting: research, theory and operations practice meet malfunctioning reality**, p. 283-288, 2004. New York, USA.

[7] COMER, Douglas E. **Internetworking With Tcp/Ip Vol. I: Principles, Protocols, And Architecture**. 4 Ed. New Jersey: Prentice Hall, 2000. 750 p. ISBN 0130183806.

[8] DAVIES, Joseph. **Understanding IPv6**. Redmond: Microsoft Press, 2002. 544 p. ISBN 0735612455.

[9] DIGERATI (Ed.). **Manual Hacker.** São Paulo: Editora Digerati, 2003. 96 p. ISBN 8589535029.

[10] DOMINGUES, Mónica. *Hosting de Websites*. **FCCN**, 2004. Disponível em: <http://www.fccn.pt/>

[11] FERRAZ, Tatiana L.; ALBUQUERQUE, Marcelo P.; ALBUQUERQUE, Márcio P. **Introdução ao Ping e Traceroute**. *Nota Técnica*. 23 f. 26 nov. 2002. Centro Brasileiro de Pesquisas Físicas, Rio de Janeiro, Brasil. CBPF-NT-010/02.

[12] FIGGINS, Stephen; SIEVER, Ellen; WEBER, Aaron. **Linux in a Nutshell**. 4 ed. O'Reilly & Associates, 2003. 944 p. ISBN 0596004826.

[13] FLICKENGER, Rob. **Linux Server Hacks**. California: O'Reilly & Associates, 2003. 240 p. ISBN 0596004613.

[14] FRIAÇAS, Carlos. *IPv6 as a basis for Future E-commerce*. **FCCN**, 2004. Disponível em: <http://www.fccn.pt/>

[15] FRIAÇAS, Carlos; FERREIRA, João. *Questões e Boas Práticas Administrativas em IPv6*. **FCCN**, 09 jul. 2004. Disponível em: <http://www.fccn.pt/>

[16] FUJISAKI, Tamohiro; KARASAWA, Kei. *Rediscovering IPv6 Part 6: Why is IPv6 IPSEC not Widely used?*. **IPv6style**, 4 jul. 2005. Disponível em: <http://www.ipv6style.jp/en/tech/20050704/index.shtml>

[17] HAENI, Reto E. **IPv6 vs. SSL: comparing Apples with Oranges**. jan. 1997, 15 f. Artigo Científico. The George Washington University Cyberspace Policy Institute, Washington DC, 1997.

[18] HAGEM, SILVIA. **IPv6 Essentials**. California: O'Reilly & Associates, 2002. 338 p. ISBN 0596001258.

[19] HANCOCK, Bill. *IPv6 Security Enhancements Still Not Everything You Need*. Discussão. **Network Security**, v. 1998, ed. 6, p. 8-10 , jun. 1998.

[20] HUITEMA, C. **IPv6: the new Internet protocol**. 2 ed. Englewood Cliffs: Prentice Hall, 1997. ISBN 0138505055.

[21] HUNGER, Steve. **Debian GNU/Linux Bible**. New York: Hungry Minds, 2001. 697 p. ISBN 0764547100.

[22] HUSTON, Geoff. **IPv4 Address Space Consuption**. *White Paper*. 3 f. mar. 2005. Disponível em < http://www.potaroo.net/papers/>

[23] KIRCH, Olaf; DAWSON, Terry. **Linux Network Administrator's Guide**. 2 ed. O'Reilly & Associates, 2000. 506 p. ISBN 1565924002.

[24] KUROSE, James F.; ROSS, Keith W. **Rede de computadores e a Internet: uma nova abordagem**. [Computer Networking: a top-down approach featuring the Internet] Trad. Arlete Simille Marques; rev. Wagner Luiz Zucchi. São Paulo: Pearson Education do Brasil, 2002. 548p. ISBN 8588639106.

[25] LAMMLE, Todd. **CCNA: Cisco Certified Network Associate Study Guide**. California: SYBEX Inc., 2005. 675p. ISBN 0782143911.

[26] LEINER, Barry M. *et al*. *A Brief History of the Internet*. **All About the Internet**. 10 dez. 2003. Disponível em: <http://www.isoc.org/internet/history/brief.shtml>

[27] LUTZ, Mark; ASCHER, David. **Learning Python**. California: O'Reilly & Associates, 1999. 368 p. ISBN 1565924649.

[28] LEWIS, Chris. **Cisco TCP/IP Routing Professional Reference**. Mcgraw-Hill, 1999. 456 p. ISBN 0070411301.

[29] MCCARTY, Bill. **Learning Debian Gnu/Linux**. O'Reilly & Associates, 1999. 360 p. ISBN 1565927052.

[30] MORTON, David. Understanding IPv6. **PC Network Advisor**. n. 83, p. 17-22, mai. 1997. Disponível em: < http://www.pcsupportadvisor.com/nasample/c0655.pdf>

[31] ODOM, Wendell. **CCNA ICND Exam Certification Guide**. Indianapolis: Cisco Press, 2004. 609 p. ISBN 158720083X.

[32] PURCELL, J. **Linux Complete Command Reference**. Indianapolis: Hed Hat Press, 1997. 1495 p. ISBN 0672311046.

[33] ROBINSON, Philip; VOGT, Harald; WAGEALLA, Waleed. **Privacy, Security and Trust within the Context of Pervasive Computing**. Boston: Springer Science + BusinessMedia, Inc., 2005. 171 p. ISBN 0387234616.

[34] SANTOS, Cleymone Ribeiro dos. **Integração de IPv6 em um Ambiente Cooperativo Seguro**. 2004. 188 f. Dissertação (Mestrado). Universidade Estadual de Campinas, Campinas, 2004.

[35] SCAMBRAY, Joel; MCCLURE, Stuart; KURTZ, George. **Hacking Exposed: Network Security Secrets & Solutions**. 2 ed. Osborne: McGraw-Hill, 2001. 703 p. ISBN 0072127481.

[36] SCHNEIER, Bruce. **Applied Cryptography: Protocols, Algorithms, and Source Code in C**. 2 ed. John Wiley & Sons, 1996. 666 p. ISBN 0471128457.

[37] SCHOLZ, Gregory R *et al*. *Internet Protocol Version 6*. **Consortium for Computing Sciences in Colleges**. *In:* **Proceedings of the seventh annual consortium for computing in small colleges central plains conference on The journal of computing in small colleges**, p. 197-204, 2001. Branson, Missouri, Estados Unidos.

[38] SIMÕES, Nelson. *Nova internet*. **Com Ciência**, 10 abr. 2002. Disponível em: <http://www.comciencia.br/reportagens/internet/net19.htm>

[39] STALLINGS, William. **Data and Computer Comunications**. 5 ed. Prentice Hall, 1996. 798 p. ISBN 0024154253.

[40] STUTZ, Michael. **The Linux Cookbook: Tips and Techniques for Everyday Use**. California: No Starch Press, 2001. 402 p. ISBN 1886411484.

[41] TANENBAUM, Andrew S. **Computer Networks.** 4 ed. Prentice Hall, 2003. 384 p. ISBN 0130661023.

[42] WEGNER, J. D.; ROCKWELL, Robert. **IP Addressing and Subnetting: including Ipv6**. Syngress Media, 2000. 487 p. ISBN 1928994016.

[43] 6BONE. Site institucional. Disponível em: <http://6bone.net>. Acesso em: 22 nov. 2004.

[44] APACHE. Site da ferramenta, [1999-2005]. Disponível em: <http://www.apache.org>. Acesso em: 09 set. 2005.

[45] ARIN. Serviços de registro, [1997-2004]. Disponível em: <http://www.arin.net>. Acesso em: 23 nov. 2004.

[46] BIND. Site da ferramenta. Disponível em: <http://www.isc.org/index.pl?/sw/bind/>. Acesso em: 09 set. 2005.

[47] BR6BONE. Site institucional. Disponível em: <http://www.6bone.rnp.br>. Acesso em: 22 nov. 2004.

[48] DEBIAN LINUX. Site da distribuição, [1997-2004]. Disponível em: <http://www.debian.org/>. Acesso em: 22 jul. 2005.

[49] DEEPSPACE6. Lista de aplicações de rede com suporte a IPv6, mantido por Peter Bieringer e outros. Disponível em: <http://www.deepspace6.net/docs/ipv6_status_page_apps.html>. Acesso em: 02 set. 2005.

[50] ETHEREAL. Site da ferramenta. Disponível em: <http://www.ethereal.com/>. Acesso em: 09 set. 2005.

[51] FIREFOX. Navegador *Web*, de código livre. Disponível em: <http://www.mozilla.org/>. Acesso em: 16 out. 2005

[52] FREENET6. Site do serviço, mantido pela HEXAGO, [1999-2004]. Disponível em: <http://www.freenet6.net>. Acesso em: 03 dez. 2004.

[53] FREES/WAN. Site do projeto. Disponível em: <http://www.freeswan.org/>. Acesso em: 09 set. 2005.

[54] IANA. Site da instituição. Disponível em: <http://www.iana.org/>. Acesso em: 22 nov. 2004.

[55] ICANN. Site da instituição. Disponível em: <http://www.icann.org/>. Acesso em: 22 nov. 2004.

[56] IETF. Site da instituição. Disponível em: <http://www.ietf.org>. Acesso em: 22 nov. 2004.

[57] IETF RFC PAGE. Site de RFCs mantido pelo IETF. Disponível em: <http://www.ietf.org/rfc.html>. Acesso em: 22 nov. 2004.

[58] IPERF. Site da ferramenta. Disponível em: <http://dast.nlanr.net/Projects/Iperf/>. Acesso em: 09 set. 2005.

[59] IPNG IMPLEMENTATIONS. Lista de organizações que estão desenvolvendo pesquisas e soluções Ipng, mantido pela Sun Microsystems. Disponível em: <http://playground.sun.com/pub/ipng/html/ipng-implementations.html>. Acesso em: 22 nov. 2004.

[60] IPV6. Site de informações sobre o protocolo IPv6. Disponível em: <http://www.ipv6.org>. Acesso em: 22 nov. 2004.

[61] IPV6 DO BRASIL. Site institucional da empresa, [2003-2004]. Disponível em: <http://www.ipv6dobrasil.com.br/>. Acesso em: 22 nov. 2004.

[62] ISOC. Site da instituição. Disponível em: <http://www.isoc.org>. Acesso em: 22 nov. 2004.

[63] KALANGO LINUX. Site da distribuição, mantido por Leandro S. dos Santos, [2003-2005]. Disponível em: <http://www.kalangolinux.org>. Acesso em: 22 jul. 2005.

[64] KAME. Projeto japonês que desenvolve suporte IPv6 e IPSEC para sistemas unix-like, em especial BSD. Disponível em: <http://www.kame.net>. Acesso em: 22 nov. 2004.

[65] KDE. Site da aplicação, mantido pela organização. Disponível em: <http://kde.org>. Acesso em: 30 ago. 2005.

[66] KERNEL LINUX. Site de desenvolvimento, mantido pela organização, [2003-2005]. Disponível em: <http://www.kernel.org>. Acesso em: 5 ago. 2005.

[67] KNOPPIX LINUX. Site da distribuição, mantido por Klaus Knopper. Disponível em: <http://www.knoppix.org>. Acesso em: 22 jul. 2005.

[68] KOMMANDER. Site da aplicação, mantido pelo time de desenvolvedores do KDE, [2005]. Disponível em: <http://kommander.kdewebdev.org>. Acesso em: 30 ago. 2005.

[69] KURUMIN LINUX. Site da distribuição, mantido por Carlos E. Morimoto, [1999-2005]. Disponível em: <http://www.guiadohardware.net/kurumin>. Acesso em: 22 jul. 2005.

[70] LACIPV6TF. Site institucional. Disponível em: <http://www.lac.ipv6tf.org>. Acesso em: 23 nov. 2004.

[71] LACNIC. Serviços de registro, [2001-2004]. Disponível em: <http://lacnic.net>. Acesso em: 23 nov. 2004.

[72] NETCAT6. Site da ferramenta. Disponível em: <http://www.deepspace6.net/projects/netcat6.html>. Acesso em: 09 set. 2005.

[73] NETFILTER. Site oficial da ferramenta *iptables*, [1999-2005]. Disponível em: <http://www.netfilter.org>. Acesso em: 28 ago. 2005.

[74] NLANR/DAST. Site Institucional. Disponível em: <http://dast.nlanr.net/>. Acesso em: 23 jul. 2005.

[75] NMAP. Site da ferramenta. Disponível em: <http://www.insecure.org/nmap/>. Acesso em: 09 set. 2005.

[76] PROJETO GNU. Site mantido pela free software foundation. Disponível em: <http://www.gnu.org>. Acesso em: 5 ago. 2005.

[77] PYTHON. Site oficial da linguagem e da fundação. Disponível em: <http://www.python.org>. Acesso em: 28 ago. 2005.

[78] PYTHON BRASIL. Site com informações sobre a linguagem. Disponível em: <http://www.pythonbrasil.com.br>. Acesso em: 28 ago. 2005.

[79] QUAGGA. Site da ferramenta. Disponível em: <http://www.quagga.net/>. Acesso em: 09 set. 2005.

[80] RADVD. Site da ferramenta. Disponível em: <http://v6web.litech.org/radvd/>. Acesso em: 09 set. 2005.

[81] RNP. Site institucional. Disponível em: <www.rnp.br>. Acesado em: 22 nov 2004.

[82] TCPDUMP. Site da ferramenta. Disponível em: <http://www.tcpdump.org/>. Acesso em: 09 set. 2005.