JOÃO GUSTAVO GAZOLLA BORGES

STUNPEDE: UM SISTEMA P2P PARA CONECTIVIDADE FIM-A-FIM TRANSPARENTE NA INTERNET USANDO TÚNEIS IPV6-SOBRE-UDP

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA

2008

JOÃO GUSTAVO GAZOLLA BORGES

STUNPEDE: UM SISTEMA P2P PARA CONECTIVIDADE FIM-A-FIM TRANSPARENTE NA INTERNET USANDO TÚNEIS IPV6-SOBRE-UDP

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA

2008

Agradecimentos

Agradeço, antes de tudo, a todas as pessoas e a todas as forças, visíveis ou invisíveis, que possibilitaram este trabalho. Agradeço principalmente ao meu orientador, prof. Elias Procópio Duarte Jr., pela grande ajuda e dedicação empreendidos na realização deste mestrado. Agradeço também a minha família pelo incondicional apoio, não só durante o mestrado, mas por todo período que o antecedeu. Não posso deixar de agradecer a meus avós, pela grande hospitalidade e por terem me agüentado tanto tempo. Agradeço aos amigos, como o Picussa e o Maverson, pelas conversas e pela motivação nas horas de desânimo. Agradeço muito também ao amigo Russo – menos conhecido como Bruno César Ribas – por ter prontamente cedido uma conta *Linux* em sua casa e liberado um porta UDP em seu *firewall*, o que possibilitou a realização dos testes. Agradeço também aos professores e funcionários da UFPR e do CNPq pelo apoio institucional e financeiro que deram a este trabalho.

Sumário

Glossário de Siglas

ACK	*Acknowledgement* (TCP)
CIDR	*Classless Inter-Domain Routing*
DCCP	*Datagram Congestion Control Protocol*
FTP	*File Transfer Protocol*
HTTP	*Hypertext Transfer Protocol*
IANA	*Internet Assigned Numbers Authority*
IP	*Internet Protocol*
IPSec	IP *Security*
IPv4	IP *version 4*
IPv6	IP *version 6*
NAPT	*Network Address Port Translator*
NAT	*Network Address Translator*
P2P	*Peer-to-Peer*
RFC	*Request for Comments*
RNP	Rede Nacional de Pesquisa
RST	*Reset* (TCP)
RTT	*Round-Trip Time*
SNMP	*Simple Network Management Protocol*
STUN	*Simple Traversal of* UDP *Through* NATs
STUNT	STUN *and* TCP *too*

SYN	*Synchronization* (TCP)
TCP	*Transmission Control Protocol*
TTL	*Time to Live*
UDP	*User Datagram Protocol*
ULA	*Unique Local* IPv6 *Unicast Addresses*
UPnP	*Universal Plug and Play*
VoIP	*Voice over Internet Protocol*
WWW	*World Wide Web*

Resumo

A adoção maciça de NAT/*firewall* na Internet afetou sensivelmente a transparência fim-a-fim da rede. Essa transparência se refere à existência de um esquema único e universal de endereçamento, assim como à idéia de conectividade fim-a-fim, isto é, a possibilidade de qualquer *host* enviar pacotes para qualquer outro *host*. NATs e *firewalls* diminuem a transparência, entre outros motivos, ao dificultarem a livre comunicação entre *hosts* internos e externos. Dois *hosts* atrás de seus respectivos NATs, por exemplo, não conseguem se comunicar utilizando o TCP/IP padrão. Para o protocolo UDP esse problema foi resolvido com o UDP *hole punching*, uma técnica que permite o estabelecimento de sessões UDP entre *hosts* atrás de NAT ou *firewall*. Para o TCP, no entanto, não existem técnicas satisfatórias. Além disso, as técnicas existentes – tanto para o UDP quanto para o TCP – são implementadas na própria aplicação, causando possível duplicação de código. Este trabalho propõe o *Stunpede*: um sistema P2P que visa permitir que *hosts* atrás de NAT ou *firewall* se comuniquem de maneira direta e transparente, utilizando qualquer protocolo de transporte. Em cada *host* do *Stunpede* é criada uma interface de rede virtual IPv6, a qual é vista por processos de aplicação como uma conexão física a uma rede IPv6. Quando um pacote IPv6 é enviado a essa interface, o *peer* local do *Stunpede* estabelece uma sessão UDP – através da Internet IPv4 – com o *peer* do *Stunpede* no *host* destinatário, por meio de UDP *hole punching*, e envia o pacote IPv6 encapsulado-o em um pacote UDP. Finalmente, o *peer* no *host* destinatário recebe o pacote e o "injeta" em sua própria interface virtual, dando a impressão de que os *hosts* estão fisicamente conectados. O *Stunpede* também utiliza *peers* "*rendezvous*" que auxiliam outros *peers* a estabelecerem sessões UDP. Estudos de caso e uma avaliação de desempenho também são apresentados.

Abstract

The massive deployment of NAT/firewall in the Internet has greatly affected its end-to-end transparency. This transparency refers to the original Internet concept of a single universal logical addressing scheme, as well as the idea of end-to-end connectivity, i.e., the possibility of any host sending packets to any other host. NATs and firewalls reduce network transparency particularly when they hinder free communication between internal and external hosts. Two hosts behind their respective NATs, for instance, cannot communicate using standard TCP/IP. Despite the existence of techniques for the establishment of UDP sessions between hosts behind NAT/firewall, the same does not hold for TCP. Furthermore, existing techniques must be implemented individually by each application, causing possible code duplication. This work proposes Stunpede: a P2P system which aims at allowing hosts behind NAT/firewall to communicate directly and transparently, over any desired transport protocol. On each host in Stunpede a virtual IPv6 network interface is created, which is viewed by local processes as a normal network interface. When an IPv6 packet is sent to that interface, the local Stunpede peer establishes an UDP session – over IPv4 Internet – with the peer in the destination host, using UDP hole punching, and sends the IPv6 packet by encapsulating it within an UDP packet. Finally, the Stunpede peer in the destination host receives the packet and "injects" it in its own virtual interface, creating the illusion that the hosts are physically connected. Stunpede also makes use of special rendezvous peers which help other peers to establish UDP sessions. Case studies and an evaluation of the system's performance are also presented.

Capítulo 1

Introdução

O grande crescimento da Internet nas últimas décadas levou ao emprego extensivo de sistemas de *firewall* [4] e NAT (*Network Address Translator* [45]), especialmente nas extremidades da rede, para amenizar problemas como segurança [3] e a escassez de endereços IPv4 públicos [20]. Em contrapartida, o uso desses dispositivos diminui sensivelmente a transparência fim-a-fim original da Internet [7]. Essa transparência se refere a um esquema de endereçamento lógico único e universal, assim como à própria conectividade fim-a-fim, isto é, a possibilidade de qualquer *host* enviar pacotes para qualquer outro *host*. NATs e *firewalls* diminuem a transparência da Internet principalmente porque esses dispositivos impedem que *hosts* externos iniciem sessões (conexões TCP ou fluxos UDP) com *hosts* internos.

Diversas aplicações, como jogos *online*, sistemas VoIP (*Voice over Internet Protocol* [16]), aplicações distribuídas e redes P2P [30], são fundamentadas na comunicação entre *hosts* situados nas extremidades da Internet, onde se concentram os NATs e *firewalls*. Como esses dispositivos dificultam o estabelecimento de sessões com os *hosts* protegidos, o uso de NAT e *firewall* causa problemas para o funcionamento dessas aplicações. Para amenizar essas dificuldades, este trabalho propõe o *Stunpede*: um sistema P2P que visa permitir a comunicação direta e transparente entre *hosts* atrás de NAT ou *firewall*. O *Stunpede* se baseia no UDP *hole punching* [11], uma técnica para o estabelecimento de sessões UDP entre *hosts* atrás de NAT/*firewall*.

Este capítulo está organizado da seguinte maneira. A seção 1.1 descreve o problema que motiva este trabalho, a seção 1.2 introduz o UDP *hole punching*, e a seção 1.3 expõe a proposta deste trabalho.

1.1 Diminuição da Transparência da Internet

O grande crescimento da Internet nas últimas décadas levou a uma adoção maciça de sistemas de *firewall* e NAT, dispositivos que dificultam a livre comunicação entre *hosts* internos e externos e por isso diminuem a transparência da rede.

Um *firewall* é uma coleção de componentes instalados estrategicamente com o objetivo de inspecionar todo o tráfego que passa entre duas redes, bloqueando ou permitindo a passagem do tráfego de acordo com um conjunto de regras, conhecido como política de segurança [43]. Na medida em que impedem a comunicação entre *hosts* internos e externos, os *firewalls* diminuem a transparência da Internet.

Um *firewall*, além de proteger redes internas contra intrusões, muitas vezes é empregado também para limitar o acesso à Internet por usuários locais ou para evitar o vazamento de informações sigilosas. Por exemplo, uma política de segurança comumente utilizada em corporações consiste em restringir severamente o tráfego de dados, de origem interna ou externa, permitindo apenas comunicações limitadas com a *Web* (WWW) por meio de um *proxy* [50]. Nesses casos, o papel do *firewall* é justamente o de diminuir a transparência da rede, limitando os serviços oferecidos aos usuários internos.

Existem situações, como em ambientes acadêmicos ou de pesquisa, por exemplo, em que um *firewall* precisa ser empregado não (tanto) para limitar o acesso à Internet e a livre troca de informações, mas principalmente para proteger a rede interna contra acesso não autorizado [17]. O uso de *firewall* nesses cenários, mesmo que com políticas pouco restritivas, limita o uso da Internet por usuários locais de maneira possivelmente não desejada, pois impede aplicações locais de receberem conexões externas ou estabelecerem conexões com outros *hosts* atrás de NAT/*firewall*, mesmo que haja o "consentimento" de ambas as partes.

Este trabalho trata das dificuldades relativas à perda de transparência causadas pelo

uso de NAT e *firewall* em redes nas quais não se deseja limitar a conectividade à Internet. Tendo isso em vista, os dispositivos de *firewall* considerados por este trabalho são os *firewalls* baseados em filtragem de pacotes – com ou sem manutenção de estado [29] – das camadas 3 e 4, isto é, que filtram pacotes inspecionando os cabeçalhos das camadas de rede e de transporte [46]. Também assume-se que a política de segurança utilizada é a de restringir que *hosts* externos ao *firewall* iniciem sessões UDP ou TCP com *hosts* protegidos, mas permitir que *hosts* internos iniciem sessões UDP ou TCP com *hosts* externos. *Firewalls* mais restritivos são geralmente utilizados para propositadamente limitar a transparência da rede e normalmente não funcionam com as técnicas apresentadas por este trabalho, e por isso não são considerados.

Um NAT é um dispositivo empregado para amenizar o problema do esgotamento de endereços IPv4 públicos que a Internet enfrenta [13]. A função de um NAT é traduzir os endereços dos pacotes que passam por ele e dessa forma interligar duas redes com domínios de endereçamento diferentes (o domínio privado e o domínio público), permitindo que *hosts* no domínio privado se comuniquem com *hosts* no domínio público. Sem a utilização de NAT, todos os *hosts* que quisessem se comunicar na Internet precisariam de um endereço público único, o que causaria o rápido esgotamento desses endereços.

Como os NATs escondem diversos endereços privados sob um único endereço público, esses dispositivos se caracterizam por bloquear sessões a *hosts* internos iniciadas por *hosts* externos. Esse comportamento faz com que o NAT funcione como uma espécie de *firewall*, protegendo a rede interna, mas tem a desvantagem de limitar a conectividade e diminuir a transparência da rede. A configuração típica de NAT/*firewall* considerada por este trabalho é baseada em estudos apresentados em [11, 18], os quais investigam as configurações de NAT/*firewall* mais utilizadas na Internet.

Devido ao fato que dispositivos de NAT e *firewall* geralmente permitem que *hosts* internos iniciem sessões com *hosts* externos, sistemas que funcionam de maneira centralizada, isto é, com poucos servidores fixos livres de NAT/*firewall*, como a *Web*, são pouco afetados pela diminuição da transparência. Isso acontece porque os clientes, mesmo que atrás de NAT ou *firewall*, conseguem se conectar normalmente aos servidores. As limitações

causadas por NATs e *firewalls* ocorrem principalmente com aplicações baseadas no paradigma P2P, que utilizam recursos das folhas da Internet e por isso precisam estabelecer sessões entre possivelmente dois *hosts* atrás de seus respectivos NATs/*firewalls*.

1.2 UDP *Hole Punching*

Uma técnica comumente utilizada para o estabelecimento de sessões UDP entre dois *hosts*, ambos atrás de NAT ou *firewall*, é o UDP *hole punching* [11]. Exemplos de aplicações que utilizam essa técnica incluem o sistema de telefonia VoIP *Skype* [2] e a aplicação de VPN *Hamachi* [42]. O UDP *hole punching* é baseado no seguinte princípio: quando um *host* interno envia um pacote UDP para a Internet, o seu dispositivo de NAT/*firewall* armazena informações sobre esse pacote que são posteriormente utilizadas para identificar pacotes de resposta e dessa forma encaminhá-los para o *host* interno correto (ou autorizar a entrada, no caso do *firewall*).

O UDP *hole punching* funciona, resumidamente, da seguinte maneira. Suponha que um *host* A deseja estabelecer uma sessão UDP com um *host* B, com ambos atrás de NAT ou *firewall*. Inicialmente, o *host* A informa ao *host* B, por meio de um servidor *rendezvous* conhecido (com o qual B mantém uma sessão UDP), que deseja iniciar uma sessão UDP com B. Em seguida, o servidor informa a B sobre a intenção de A em estabelecer uma sessão, e os *hosts* começam a enviar pacotes UDP um para o outro, até que uma resposta seja recebida. O envio de um pacote UDP de A para B, por exemplo, cria um "furo" no NAT/*firewall* de A, o qual é utilizado pelo *host* B para "penetrar" o NAT/*firewall* do *host* A; e vice-versa. Para funcionar, cada *host* precisa conhecer o *endpoint* – isto é, o par ⟨IP, porta⟩ – público do outro, o qual é descoberto pelo *rendezvous* e informado a cada *host*. Uma vez terminado o processo, os *hosts* podem se comunicar normalmente, utilizando o protocolo UDP.

Também existem soluções, chamadas de TCP *hole punching* [11], que permitem o estabelecimento de conexões TCP entre *hosts* atrás de NAT/*firewall*, utilizando o mesmo princípio básico do UDP *hole punching*. Em contrapartida, o TCP *hole punching* é mais complexo e assume diversas condições que nem sempre são satisfeitas, fazendo com que o

TCP *hole punching* seja menos robusto que o UDP *hole punching* [18].

1.3 Proposta deste Trabalho

Os métodos de passagem de NAT e *firewall* apresentados anteriormente precisam ser implementados individualmente por qualquer aplicação que queira estabelecer comunicações entre *hosts* atrás desses dispositivos. Além disso, como o TCP *hole punching* é mais complexo e menos robusto que o UDP *hole punching*, aplicações que o utilizem são menos confiáveis.

Este trabalho propõe o *Stunpede*, um sistema P2P que visa permitir que processos de aplicação situados em *hosts* atrás de NAT ou *firewall* se comuniquem de maneira direta e transparente, utilizando qualquer protocolo de transporte. O sistema proposto funciona da seguinte maneira. Um *peer* do *Stunpede* é executado em cada *host* participante. Esse *peer* é responsável por criar uma interface de rede virtual IPv6 em seu *host*, a qual é vista por aplicações locais como uma interface normal. Quando um pacote IPv6 é enviado a essa interface, o *peer* local estabelece uma sessão UDP – através da Internet IPv4 – com o *peer* do *Stunpede* no *host* destinatário, por meio de UDP *hole punching*, e envia o pacote IPv6 encapsulado-o em um pacote UDP. Finalmente, o *peer* no *host* destinatário recebe o pacote e o "injeta" em sua própria interface virtual, dando a impressão de que os *hosts* estão fisicamente conectados.

O *Stunpede* também emprega *peers* especiais, chamados de *rendezvous*, os quais ajudam *peers* atrás de NAT ou *firewall* a estabelecerem sessões UDP. Além disso, todo *peer* no *Stunpede* mantém uma sessão UDP ativa com um *rendezvous*, o qual tem a função de avisar um *peer* cliente de que outro *peer* deseja estabelecer uma sessão UDP.

O motivo da escolha do IPv6 se deve ao grande espaço de endereçamento desse protocolo e, em especial, à existência dos endereços locais do tipo ULA (*Unique Local* IPv6 *Unicast Addresses* [22]), utilizados nas interfaces virtuais. O endereço IPv6 de cada *host* é utilizado para identificar o servidor de *rendezvous* ao qual o *host* (*peer*) está conectado. Isso é feito codificando o endereço IPv4 e a porta UDP do servidor de *rendezvous* dentro do endereço IPv6 do *host*. Esse mecanismo permite que o servidor *rendezvous* de um *peer*

seja determinado imediatamente, bastando conhecer seu endereço IPv6.

A solução P2P é proposta para permitir que o sistema seja escalável, em comparação à utilização de um servidor central. Os *rendezvous* não afetam o desempenho do *Stunpede*, pois a busca por um *rendezvous* é feita instantaneamente. Além disso, como os *rendezvous* não se comunicam entre si, eles representam uma maneira simples e eficiente de distribuir a carga do sistema.

1.4 Organização deste Trabalho

Este trabalho está organizado da seguinte maneira. O capítulo 2 analisa os princípios de transparência presentes na arquitetura original da Internet e descreve o funcionamento de dispositivos de NAT e *firewall*, os principais responsáveis por abalar esses princípios. O capítulo 3 expõe soluções da literatura para contornar a falta de transparência da Internet. O capítulo 4 apresenta a proposta deste trabalho, os estudos de caso e uma avaliação do sistema. O capítulo 5 conclui o trabalho.

Capítulo 2

Transparência da Internet

O modelo original do protocolo de rede da Internet (*Internet Protocol* [36, 46]) era caracterizado por uma grande transparência fim-a-fim [7]. Essa transparência se refere a um esquema único e universal de endereçamento, assim como ao princípio de conectividade fim-a-fim, isto é, a possibilidade de qualquer *host* enviar pacotes para qualquer outro *host*, de maneira desimpedida. Outro aspecto da transparência é a propriedade de os pacotes trafegarem na rede, da origem ao destino, essencialmente inalterados. Em contrapartida, a Internet vem enfrentando uma progressiva perda de transparência, o que faz com que a rede funcione de maneira diversa do que foi projetada e do que seria ideal para as aplicações.

Este capítulo está organizado da seguinte maneira. A seção 2.1 analisa os princípios de transparência da arquitetura original da Internet. As seções 2.2 e 2.3 descrevem o funcionamento de sistemas de NAT e *firewall*, respectivamente, e discutem como esses dispositivos diminuem a transparência da Internet.

2.1 Transparência Fim-a-Fim

A Internet vem sofrendo crescentes dificuldades com a diminuição da transparência nas comunicações fim-a-fim. Esse fato, conhecido como "o problema fim-a-fim" (*the end-to-end problem*), é discutido em diversos trabalhos [6, 7, 25, 37, 41]. A transparência nas comunicações fim-a-fim se refere, entre outros princípios, a um esquema de endereçamento

globalmente único, no qual cada endereço IP representa de maneira inequívoca um único *host*. A transparência fim-a-fim também está relacionada à propriedade de os pacotes IP trafegarem pela rede essencialmente inalterados, a qual é fundamental para o funcionamento de protocolos como o IPSec [26]. Outro aspecto importante da transparência é a conectividade fim-a-fim, isto é, a noção de uma rede representável por um grafo conexo na qual cada *host* pode enviar mensagens livremente para qualquer outro *host* cujo endereço seja conhecido, sem nenhum bloqueio.

Os principais responsáveis pela diminuição da transparência da Internet são os sistemas de *firewall* e NAT. Um *firewall*, devido à natureza inerente de bloquear tráfego entre duas redes, abala o princípio da conectividade fim-a-fim, pois impede a livre comunicação entre os *hosts* internos e externos. Um NAT é um dispositivo instalado entre duas redes com domínios de endereçamento diferentes, com a função de alterar (traduzir) o cabeçalho dos pacotes IP que passam pelo NAT, permitindo a interligação das redes. Como por definição um NAT altera os pacotes IP, esse dispositivo compromete o princípio de os pacotes trafegarem inalterados pela rede. Esse comportamento interrompe o funcionamento de protocolos de segurança que verificam a integridade dos pacotes. Além disso, *hosts* externos geralmente não conseguem iniciar sessões com *hosts* atrás de NAT, o que contribui para a diminuição da conectividade fim-a-fim da Internet.

2.1.1 O Argumento Fim-a-Fim

Outro aspecto da arquitetura da Internet é o argumento fim-a-fim (*the end-to-end argument*), apresentado em [6, 7, 41]. Segundo esse argumento, como princípio básico, certas funções fim-a-fim só podem ser corretamente executadas pelos próprios sistemas finais, mas não pelo canal de comunicação. Um exemplo é a função de garantir a integridade da comunicação. Mesmo que uma rede ofereça confiabilidade total, falhas em outros componentes podem ocorrer, como disco ou memória, e por isso o argumento sugere que é melhor dar a responsabilidade de garantir a integridade aos sistemas finais, os quais precisam realizar essa função de qualquer maneira.

Por exemplo, considere uma aplicação de transferência de arquivos que precisa ser bas-

tante robusta. Mesmo que o canal de comunicação oferecesse um serviço 100% confiável, garantia que pode ser custosa, a aplicação ainda tem que realizar verificações de integridade para detectar falhas não relacionadas ao canal de comunicação, como falhas transientes de memória ou de leitura em disco. Em outras palavras, mesmo que um grande esforço no meio de comunicação seja feito para garantir uma confiabilidade perfeita, os sistemas finais ainda teriam o ônus de fazer suas próprias verificações. Por causa disso, o argumento sugere que a responsabilidade da integridade seja atribuída aos sistemas finais. No entanto, o argumento ressalva que certa garantia de confiabilidade pode ser implementada no canal de comunicação, por questões de desempenho, mas nunca uma garantia que vise a perfeição, a qual pode ser custosa e de pouca utilidade para os pontos finais.

No caso específico da transparência fim-a-fim da Internet, esse princípio tem conseqüências importantes quando se deseja que aplicações sejam tolerantes a falhas parciais na rede. Segundo o argumento, um protocolo ponto a ponto (como o TCP, por exemplo) não pode ser projetado de maneira que informações de estado (como o estado de uma conexão TCP) fiquem armazenadas dentro da rede, mas somente nos pontos finais. Se esse princípio for obedecido, o estado da comunicação só é destruído se um dos pontos finais falhar, e falhas temporárias no canal de comunicação podem ser toleradas.

No entanto, NATs e *firewalls* em geral guardam informações sobre o estado de conexões TCP e de sessões UDP internamente, e essas informações são fundamentais para a comunicação entre *hosts* atrás desses dispositivos com outros *hosts* na Internet. Se um sistema de NAT ou *firewall* sofrer uma falha e perder esses estados, as conexões ativas que passavam pelo NAT ou *firewall* serão perdidas, o que torna esses dispositivos pontos únicos de falha e diminui a confiabilidade que é esperada em uma rede baseada em comutação de pacotes, como a Internet.

2.2 NAT - *Network Address Translator*

Esta seção descreve o funcionamento dos sistemas de NAT, dispositivos comumente usados que contribuem para a diminuição da transparência da Internet. O entendimento desses

dispositivos se faz mister para o desenvolvimento de aplicações que consigam operar adequadamente no cenário atual da Internet, caracterizado pela falta de transparência.

2.2.1 Exaustão dos Endereços IPv4

O protocolo da camada de rede da Internet, o IPv4 (*Internet Protocol version 4* [36]), define um espaço fixo de 32 bits para os endereços IP. Desde o projeto inicial do IPv4, alterações têm sido feitas na semântica dos endereços IP numa tentativa de minimizar o rápido esgotamento desses endereços.

Originalmente, um endereço IP era dividido em dois pedaços lógicos: os primeiros 8 bits identificavam uma rede e os 24 bits restantes um *host* dessa rede. Como esse esquema comporta a existência de 256 redes com 16.777.216 de *hosts* cada, logo se percebeu a necessidade de uma maior flexibilidade, pois qualquer rede, independente do tamanho, consome 1/256 de todo espaço de endereçamento. A solução foi definir 3 classes de endereços IP (A, B, C) para endereçamento *unicast* e 2 classes (D, E) para usos especiais. Um endereço classe A, B ou C também é dividido em dois pedaços lógicos – rede e *host*, mas a quantidade de bits de cada parte varia de classe para classe, conforme mostra a tabela 2.1. A vantagem desse novo esquema é sua maior flexibilidade, uma vez que redes pequenas podem utilizar fatias menores do espaço de endereçamento.

O esquema de classes também se mostrou inflexível [14]. As organizações de modo geral consideravam um bloco da classe C, com 254 endereços, muito pequeno, e por isso davam preferência a blocos da classe B, causando a rápida exaustão dessa classe. Além disso, um bloco de endereços da classe B, com capacidade para 65534 *hosts*, é geralmente muito grande para organizações de tamanho médio e acarreta grande desperdício de endereços. Para tornar o endereçamento do IPv4 mais flexível, o esquema *Classless Inter-Domain Routing* (CIDR) foi adotado [38, 14, 12, 23, 49], o qual é utilizado atualmente.

A julgar pelo nome, o método CIDR não utiliza o conceito de classes. A idéia central desse esquema é fazer com que a divisão do endereço IP nas partes de rede e *host* seja especificada para cada endereço, por meio de um número que indica quantos bits do endereço fazem parte da rede. O endereço 200.17.202.17/8, por exemplo, representa o

Classe	Bits Iniciais	Bits de Rede (# de Redes)	Bits de *Host* (# de *Hosts*)
A	0	7 (128)	24 (16.777.216)
B	10	14 (16.384)	16 (65.536)
C	110	21 (2.097.152)	8 (256)

Tabela 2.1: Classes de Endereços IP.

endereço 200.17.202.17 com os 8 primeiros bits identificando a rede. A principal vantagem do CIDR é a grande granularidade com que os endereços podem ser distribuídos [48, 46]. Uma rede com 2 *hosts*, por exemplo, poderia receber o bloco 200.17.202.252/30, o qual possui apenas 4 endereços.

Na década de 1990, com o grande crescimento da Internet, constatou-se que o CIDR não seria suficiente para conter o esgotamento dos endereços. Em face dessa questão, duas soluções foram consideradas: o uso de NAT para reaproveitar os endereços IP e aumentar a sobrevida do IPv4; e a criação de um novo protocolo, o IPv6, com um espaço de endereçamento exorbitantemente grande. O uso de NAT foi tão bem sucedido em frear o consumo dos endereços IPv4 que a adoção global do IPv6 se tornou uma incógnita.

2.2.2 NAT – Visão Geral

O uso de NAT [9] corresponde à solução encontrada, na década de 1990, para reaproveitar os endereços IPv4 existentes. Primeiramente, dividiu-se o espaço de endereçamento do IPv4 em dois domínios: um domínio com endereços públicos, globalmente únicos por toda a Internet, e outro com endereços privados, não únicos e não roteáveis na Internet global. Para fazer a divisão, a entidade que gerenciava as alocações de endereço IP na Internet – a IANA (*Internet Assigned Numbers Authority*) – reservou três faixas de endereços (10.0.0.0/8, 172.16.0.0/12 e 192.168.0.0/16) para serem utilizados somente em redes privadas [39]. Como um mesmo endereço privado pode ser utilizado por diversos *hosts* situados em redes locais diferentes, esses endereços são reutilizados, poupando endereços públicos.

Para permitir que *hosts* com endereços privados se comuniquem com *hosts* na Internet é necessário o emprego de NAT: um dispositivo que tem a função de traduzir os endereços dos pacotes que passam por ele, possibilitando a interligação de duas redes com domínios

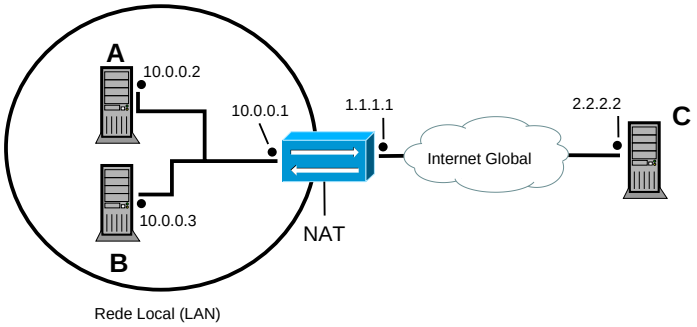

Figura 2.1: Configuração de rede comumente utilizada por um NAT.

de endereçamento diferentes. Em uma configuração típica, ilustrada na 2.1, um NAT possui duas interfaces de rede: uma interface com endereço público, conectada à Internet global, e uma interface com endereço privado, conectada a uma rede local, na qual situam-se um ou mais *hosts* com endereços privados. Quando um pacote IP é enviado da rede local para a Internet, o NAT troca o endereço de origem do pacote, um endereço privado, para seu endereço público, e envia o pacote como se o próprio NAT fosse o remetente. Os NATs também comumente alteram a porta UDP ou TCP de origem do pacote, para permitir que várias sessões com a mesma porta de origem sejam abertas por *hosts* internos. Além disso, um NAT também mantém controle sobre dados básicos de cada sessão ativa entre um *host* interno e um *host* externo. Quando uma resposta a um pacote enviado para a Internet é recebida, o NAT utiliza os dados sobre as sessões para realizar a tradução inversa. A tradução inversão é feita trocando o endereço e a porta de destino do pacote para o endereço privado e a porta local do *host* que deve recebê-lo.

2.2.3 Vantagens e Desvantagens

A utilização de NAT é uma solução simples e de baixo custo que se mostrou bastante eficaz na redução do consumo de endereços IP públicos. Além disso, a presença de NAT é transparente para aplicações que não precisam receber conexões externas. Quando o NAT foi inicialmente adotado na Internet, a maioria das aplicações seguia o paradigma cliente/servidor, e por isso não tinham problemas com NAT.

Entretanto, o uso de NAT possui diversas desvantagens. A principal é que, como *hosts* atrás de NAT possuem endereços IP sem significado na Internet, esses *hosts* não são

endereçáveis diretamente e não podem escutar por conexões externas, o que prejudica a conectividade fim-a-fim da rede.

Além de diminuir a conectividade, um NAT também diminui a transparência da Internet quando altera os pacotes IP. Essa característica faz com que sistemas de segurança que verificam a integridade das comunicações deixem de funcionar. Um desses sistemas é o IPSec [26], uma coleção de protocolos que adiciona segurança ao protocolo IP. O IPSec possui um mecanismo de segurança chamado *Authentication Header* [1], o qual funciona gerando um *hash* criptográfico com todo o cabeçalho IP, antes de um pacote ser enviado. Esse *hash* é então utilizado pelo recipiente para autenticar o pacote. Se o cabeçalho do pacote original for modificado, como fazem os NATs, a autenticação falha e o recipiente tem que descartar o pacote [33].

2.2.4 Classificação e Funcionamento

Esta seção classifica os NATs com base em [45, 40] e apresenta uma descrição mais detalhada do funcionamento desses dispositivos. Existem dois tipos de NAT: NAT básico e NAT com tradução de portas. Um NAT do primeiro tipo, mais simples, é chamado NAT básico (*basic* NAT) ou *one-to-one* NAT. Um NAT básico traduz apenas endereços IP, e suporta um número limitado de *hosts* internos simultaneamente se comunicando com *hosts* externos. Um NAT do segundo tipo, chamado NAT com tradução de portas (NAPT - *Network Address Port Translation*), ou apenas NAT, traduz portas TCP/UDP além de endereços IP, e permite que milhares de *hosts* internos se comuniquem com *hosts* externos simultaneamente, com um único endereço público.

Como os NATs do tipo NAPT são mais comuns na Internet e abrangem o funcionamento dos NATs básicos, apenas os NATs com tradução de porta (NAPT) são descritos nesta seção, os quais são designados neste texto apenas por "NAT".

Um NAT possui pelo menos duas interfaces de rede: uma interface com endereço privado (*ip_interno*), conectada a uma rede local, e uma interface com endereço público (*ip_externo*), conectada à Internet global. Um NAT funciona como um roteador, enviando pacotes entre duas redes. Além disso, um NAT também possui uma tabela interna

que é utilizada para manter controle sobre todas as sessões ativas iniciadas por *hosts* internos. Cada entrada nessa tabela armazena informações sobre uma sessão ativa, e possui normalmente os seguintes campos:

1. Tipo da sessão, como UDP ou TCP, por exemplo.
2. Endereço IP local (*ip_local*): endereço IP, utilizado como endereço de origem pelo *host* interno que iniciou a sessão.
3. Porta TCP/UDP local (*porta_local*): porta local utilizada como porta de origem pelo *host* interno para enviar os pacotes dessa sessão.
4. Endereço IP de destino (*ip_destino*) e porta TCP/UDP de destino (*porta_destino*): endereço IP e porta de destino da sessão.
5. Porta TCP/UDP externa do NAT (*porta_externa*): corresponde à porta associada pelo NAT em sua interface pública, para utilizar como porta de origem dos pacotes desta sessão, ao enviá-los para a Internet.
6. Temporizador: marca o tempo decorrido desde o último pacote recebido pelo NAT, relativo a esta sessão. Se o temporizador atingir um determinado limite, o NAT considera que a sessão terminou e a elimina da tabela.

Quando um *host* interno envia um pacote IP para a Internet, o NAT recebe o pacote e realiza os seguintes passos:

1. Verificar se o pacote corresponde a uma sessão ativa, analisando o cabeçalho do pacote e pesquisando em sua tabela de sessões.
    - Em caso negativo (o pacote não pertence a nenhuma sessão ativa):
        (a) Determinar uma porta externa (*porta_externa*), para ser utilizada como porta de origem para enviar os pacotes relativos a esta sessão para a Internet. O método utilizado para determinar essa porta é descrito posteriormente neste capítulo.

(b) Criar uma nova entrada na tabela, adicionando informações sobre a sessão recém criada.

- Em caso afirmativo (o pacote pertence a uma sessão ativa):

  (a) Atualizar o temporizador para a sessão, na tabela de sessões.

2. Traduzir o pacote, isto é, alterar o endereço e a porta de origem do pacote antes de enviá-lo para a Internet. Mais especificamente, o NAT traduz o endereço IP de origem do pacote para o endereço externo do NAT (*ip_externo*). O NAT também altera a porta de origem do pacote para a porta externa do NAT (*porta_externa*), de acordo com o valor desse campo na tabela de sessões.

3. Como o NAT alterou o cabeçalho do pacote, o *checksum* do cabeçalho IP precisa ser recalculado. Além disso, como portas TCP/UDP foram alteradas, é preciso recalcular também o *checksum* desses protocolos.

4. Enviar o pacote para a Internet.

Quando um NAT recebe um pacote da Internet, o NAT utiliza sua tabela de sessões para determinar o *host* interno ao qual a mensagem deve ser encaminhada e para saber como traduzir o pacote. Para isso o NAT realiza os seguintes passos:

1. Verificar se o pacote corresponde a uma sessão ativa, analisando o cabeçalho do pacote e pesquisando em sua tabela de sessões. Em caso negativo, o pacote é descartado, pois o NAT não tem como saber para qual *host* interno enviá-lo.

2. Em caso afirmativo, traduzir o pacote, isto é, alterar o endereço e a porta de destino do pacote, antes de enviá-lo ao *host* interno correto. Mais especificamente, o NAT traduz o endereço IP de destino do pacote para o endereço do *host* interno que deve recebê-lo. O NAT também altera a porta de destino do pacote para a porta local do *host* interno (*porta_local*), conforme informações da tabela de sessões.

3. Recalcular o *checksum* do IP e do TCP/UDP.

4. Enviar o pacote para a rede local, para o *host* correspondente à sessão.

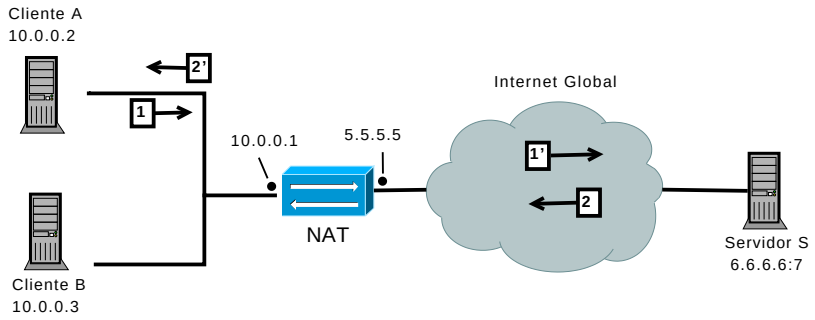

Figura 2.2: Cenário de NAT.

Como exemplo, considere o cenário mostrado na figura 2.2. O dispositivo de NAT possui o IP público 5.5.5.5. O *host S* possui o endereço público 6.6.6.6 e executa um servidor de *echo* [35], escutando na porta UDP 7. O servidor de *echo* simplesmente manda de volta para o cliente qualquer pacote UDP que receber, sem alterações. Suponha que o NAT foi reinicializado, isto é, que sua tabela de sessões está vazia.

Considere que o cliente *A* envia um pacote UDP para o servidor de *echo*, utilizando a porta 2048 como porta local UDP. Esse pacote é representado pelo número 1 na figura 2.2, e seu cabeçalho básico pode ser visto a seguir:

Cabeçalho IP	Source Address: 10.0.0.2 Destination Address: 6.6.6.6	
Cabeçalho UDP	Source Port: 2048	Destination Port: 7
Dados	...	

Ao receber o pacote, o NAT verifica que não existe nenhuma sessão ativa relacionada, pois sua tabela de sessões está vazia. Em seguida, o NAT escolhe uma porta externa de origem, como por exemplo 4040, e traduz tanto o endereço como a porta de origem do pacote. O NAT também cria uma nova entrada em sua tabela, para manter controle sobre a sessão recém criada. A tabela de sessões do NAT fica assim:

#	tipo	*ip_local*	*porta_local*	*ip_externo*	*porta_externa*	*ip_dst*	*porta_dst*
1	UDP	10.0.0.2	2048	5.5.5.5	4040	6.6.6.6	7

Finalmente, o NAT envia o pacote traduzido para a Internet, o qual é representado na figura 2.2 pelo número 1'. O pacote traduzido fica assim (os campos em negrito representam alterações em relação ao pacote antes de ser traduzido):

Cabeçalho IP	Source Address: **5.5.5.5** Destination Address: 6.6.6.6	
Cabeçalho UDP	Source Port: **4040**	Destination Port: 7
Dados	...	

Em seguida, o servidor S recebe o pacote normalmente, como se o NAT de A fosse o verdadeiro remetente. O servidor de *echo* envia então um pacote UDP de resposta, contendo o conteúdo (*payload*) do pacote recebido. Esse pacote é mostrado pelo número 2 na figura 2.2, e possui o seguinte cabeçalho:

Cabeçalho IP	Source Address: 6.6.6.6 Destination Address: 5.5.5.5	
Cabeçalho UDP	Source Port: 7	Destination Port: 4040
Dados	...	

O NAT, ao receber a resposta, verifica em sua tabela que existe uma sessão ativa correspondente ao pacote. Dessa forma, o NAT pode traduzir o pacote de maneira inversa, isto é, gerando o seguinte pacote (mostrado pelo número 2' na figura 2.2):

Cabeçalho IP	Source Address: 6.6.6.6 Destination Address: **10.0.0.2**	
Cabeçalho UDP	Source Port: 7	Destination Port: **2048**
Dados	...	

Em seguida, o cliente A recebe o pacote 2', como se não houvesse nenhum NAT entre ele e o servidor. Ou seja, em casos como esse, o uso de NAT é transparente para a aplicação.

O funcionamento apresentado corresponde a um *modelo* geral dos NATs, apresentado para dar uma visão geral. A operação detalhada desses dispositivos não é padronizada, o que dificulta o funcionamento das técnicas de passagem de NATs, apresentadas no próxima capítulo. Para melhor entender os diferentes comportamentos encontrados nos dispositivos, com relação aos pacotes UDP os NATs podem ser divididos em 2 subtipos básicos [40] – NAT *cone* e NAT simétrico – descritos a seguir. Um estudo do comportamento dos NATs em relação ao TCP pode ser encontrado em [18].

2.2.4.1 NAT *Cone*

Um NAT *cone* é um NAT no qual todas as requisições provenientes do mesmo endereço IP e da mesma porta local são mapeadas para a mesma porta externa (pública), independente do endereço de destino. Por exemplo, se um *host* interno iniciar uma sessão enviando um pacote UDP com porta de origem p_{orig} e porta de destino $p1_{dest}$ para determinado *host* w na Internet, uma porta externa p_{ext} é escolhida pelo NAT, e o pacote é enviado para a Internet com porta de origem p_{ext}. Se, enquanto essa sessão estiver ativa, o mesmo *host* interno enviar outro pacote com a mesma porta de origem p_{orig} mas para para outro *host* z ($z \neq w$), a mesma porta pública p_{ext} é utilizada pelo NAT.

Retomando o exemplo apresentado anteriormente (vide figura 2.2), considere que o cliente A já tenha recebido a resposta do servidor S, mas que a sessão entre A e S ainda esteja ativa (ou seja, o temporizador do NAT para a sessão ainda não expirou). Suponha que o cliente A envia outro pacote UDP, utilizando a mesma porta de origem utilizada ao enviar o pacote para S, isto é, a porta 2048, mas agora envia para outro servidor, T, que está escutando em uma porta qualquer. Supondo que o endereço IP de T é 200.10.10.10, e que T escuta na porta 3000, o pacote enviado por B é o seguinte:

Cabeçalho IP	Source Address: 10.0.0.2 Destination Address: 200.10.10.10	
Cabeçalho UDP	Source Port: 2048	Destination Port: 3000
Dados	...	

Quando esse pacote chega ao NAT, este verifica que existe uma sessão em sua tabela cujo endereço e porta de origem é o mesmo do pacote, isto é, $\langle$ *ip_local* = 10.0.0.2, *porta_local* = 2048 $\rangle$. No entanto, como o endereço e a porta de destino são diferentes, este pacote corresponde a uma nova sessão, que é acrescentada na tabela de sessões. Se o NAT de A é do tipo *Cone*, o NAT irá associar a mesma porta externa, 4040, para enviar os pacotes para T. Com isso, a tabela de sessões do NAT fica assim:

#	tipo	*ip_local*	*porta_local*	*ip_externo*	*porta_externa*	*ip_dst*	*porta_dst*
1	UDP	10.0.0.2	2048	5.5.5.5	4040	6.6.6.6	7
2	UDP	10.0.0.2	2048	200.10.10.10	4040	6.6.6.6	3000

Em outras palavras, em um NAT *Cone*, para quaisquer duas entradas na tabela

de sessões do NAT, a e b, se $ip\_local_a = ip\_local_b \wedge porta\_local_a = porta\_local_b$, então $porta\_externa_a = porta\_externa_b$.

De acordo com as regras de aceitação de pacotes UDP externos, os NATs do tipo *Cone* costumam ser subdivididos em 3 tipos:

Full Cone Em NATs desse tipo, enquanto uma sessão qualquer estiver ativa entre um *host* interno e um *host* externo, sendo i a porta externa associada pelo NAT para a sessão, qualquer *host* na Internet, utilizando qualquer porta de origem, pode enviar pacotes para o *host* interno, bastando enviar um pacote para a porta i, no endereço público do NAT.

Por exemplo, após o NAT ter enviado o pacote para T, suponha que um terceiro servidor, Y, com IP 189.3.3.3 e porta local 25, tenta enviar o seguinte pacote para a porta 4040 do NAT:

Cabeçalho IP	Source Address: 189.3.3.3 Destination Address: 5.5.5.5	
Cabeçalho UDP	Source Port: 25	Destination Port: 4040
Dados	...	

Quando o NAT recebe esse pacote, ele verifica que o pacote não pertence a nenhuma sessão ativa. Entretanto, o NAT constata que existe pelo menos uma sessão ativa que utiliza a porta 4040 como porta externa. Se o NAT for do tipo *Full Cone*, esse NAT irá permitir a passagem do pacote para o *host* A, mesmo não pertencendo a nenhuma sessão ativa. No entanto, o NAT pode ou não adicionar uma nova entrada em sua tabela, dependendo de sua implementação. O pacote traduzido pelo NAT, e enviado para o *host* interno, é o seguinte:

Cabeçalho IP	Source Address: 189.3.3.3 Destination Address: 10.0.0.2	
Cabeçalho UDP	Source Port: 25	Destination Port: 2048
Dados	...	

Cone Restricted Um NAT do tipo *cone restricted* é similar ao NAT *full cone*, com a diferença básica de que um *host* externo x só pode enviar um pacote para um *host*

interno se houver pelo menos uma sessão ativa no NAT entre o *host* interno e o *host* x. Por exemplo, se o NAT da figura 2.2 é desse tipo, quando o NAT recebe o pacote de T, o NAT verifica que não existe nenhuma sessão ativa entre A e T e descarta o pacote.

Port Restricted Cone Um NAT do tipo *port restricted cone* é similar ao NAT *cone restricted*, mas a restrição também inclui número de portas. Em outras palavras, um *host* externo x só pode enviar um pacote, com porta de origem p, para um *host* interno y, se y tiver iniciado anteriormente uma sessão UDP com o *host* x na porta p. Em outras palavras, um *host* externo só pode mandar um pacote para um *host* interno se o pacote for rigorosamente relacionado a uma sessão ativa iniciada por um *host* interno.

2.2.4.2 NAT Simétrico

Em um NAT simétrico, cada sessão ativa possui uma porta externa diferente, e um *host* externo x, utilizando a porta local p, só pode enviar um pacote para um *host* interno se houver uma sessão ativa inicia por um *host* interno com o *host* x, na porta p.

2.3 *Firewalls*

Nos primórdios da Internet havia uma comunidade pequena de usuários confiáveis que prezavam por princípios como a total abertura da rede, o compartilhamento e a colaboração [24]. No entanto, com o crescimento da rede, a manutenção desses princípios ficou cada vez mais difícil, devido ao grande risco que o acesso ilimitado às redes conectadas representava. Em 1988, um *worm* que explorava falhas em programas utilitários em sistemas baseados no UNIX-BSD infectou milhares de máquinas e interrompeu o funcionamento da Internet por vários dias [44]. O *Morris Worm*, como ficou conhecido, abalou a idéia de uma Internet aberta e confiável. Além desse incidente, diversas intrusões ou tentativas foram registradas na mesma época [47, 8].

Na década de 1990, com o grande crescimento da quantidade de invasões, ficou cada vez mais clara a necessidade de se proteger as redes conectadas contra ataques vindos da Internet. Em [3], por exemplo, Steve Bellovin descreve uma série de ataques descobertos

naquela época, a partir da monitoração das redes da AT&T, o que tornou evidente a existência de usuários maliciosos na Internet e a crônica falta de segurança da mesma.

Devido à necessidade de barrar o acesso indevido a uma rede, foram desenvolvidos dispositivos de proteção, chamados *firewalls*. Uma definição de *firewall* é apresentada em [24]: "um *firewall* é uma máquina ou coleção de máquinas entre duas redes, que obedecem os seguintes critérios:

- O *firewall* é instalado nas bordas entre as duas redes;
- Todo o tráfego entre as duas redes deve passar pelo *firewall*;
- O *firewall* possui um mecanismo que permite a passagem de certos tráfegos e o bloqueio de outros. As regras descrevendo esse mecanismo correspondem à política do *firewall*."

O uso de *firewalls* se deve principalmente à pouca segurança oferecida pela maioria dos sistemas operacionais. Se uma organização permitir acesso ilimitado à sua rede interna, a partir da Internet, a rede interna só pode ser considerada segura se os serviços de rede assim como os sistemas operacionais de todas as máquinas internas, individualmente, forem completamente seguros. No entanto, proteger completamente todos os computadores de uma rede é uma tarefa constante e complexa, e por isso muitas organizações optam por, ao invés de deixar os computadores internos individualmente protegidos, concentrar todos os esforços na implementação de um sistema de *firewall* que proteja a rede inteira [24]. No entanto, essa abordagem possui o risco de que se uma máquina interna for comprometida, toda a rede fica vulnerável [32].

2.3.1 Classificação

Com base em [4], esta sessão descreve os dois principais tipos de *firewall*: os *firewalls* da camada de aplicação e os *firewalls* baseados em filtragem de pacotes.

2.3.1.1 *Firewalls* da Camada de Aplicação

Os *firewalls* da camada de aplicação operam inspecionando e bloqueando mensagens na camada de aplicação da pilha de protocolos TCP/IP. Como existem diversos protocolos de aplicação, *firewalls* desse tipo precisam saber como interpretar o protocolo de cada aplicação que será permitida na rede.

Como inspecionam todo o conteúdo das mensagens, *firewalls* desse tipo são considerados bastante seguros. Por exemplo, é possível filtrar mensagens HTTP que possuam determinada palavra, ou requisições FTP que solicitem por um arquivo específico. No entanto, devido à grande diversidade de protocolos de aplicação, a implementação de um *firewall* de aplicação não é uma tarefa trivial.

2.3.1.2 *Firewalls* Baseados em Filtragem de Pacotes

Firewalls baseados em filtragem de pacotes atuam na camada de rede da pilha de protocolos TCP/IP, isto é, na camada IP. Esses *firewalls* comumente também inspecionam cabeçalhos da camada de transporte, como portas TCP/UDP. Um *firewall* baseado em filtragem de pacotes funciona analisando individualmente o cabeçalho de cada pacote que passa por ele, como endereços e portas de origem e destino, para determinar se o pacote é aceito na rede ou descartado. O administrador do *firewall* determina um conjunto de regras que é utilizado pelo *firewall* para decidir o que fazer com cada pacote.

Firewalls baseados em filtragem de pacotes podem ser divididos em duas categorias: *firewalls* com estado (*stateful*) e *firewalls* sem estado (*stateless*). *Firewalls* sem estado inspecionam cada pacote individualmente, e tomam a decisão de aceitar ou descartar o pacote com base unicamente em dados do próprio pacote, considerado isoladamente. Em outras palavras, um *firewall* sem estado não leva em conta o contexto do pacote, como por exemplo se ele pertence a uma sessão ativa ou não. Por definição, *firewalls* sem estado não armazenam informações de estado sobre sessões ativas entre *hosts* internos e *hosts* externos, e por isso são bastante simples.

Por outro lado, *firewalls* com estado mantêm informações de estado sobre cada sessão ativa entre um *host* interno e um *host* externo, como conexões TCP ou sessões UDP. Por

exemplo, quando um *host* interno inicia uma conexão TCP com um *host* externo, o *firewall* armazena internamente dados sobre essa conexão, como endereços e portas de origem e destino, assim como o estado da conexão. *Firewalls* com estado também inspecionam cada pacote individualmente, assim como os *firewalls* sem estado. Apesar disso, um *firewall* com estado pode utilizar, para determinar se um pacote é aceito ou descartado, também informações sobre o contexto do pacote, isto é, sobre uma sessão à qual o pacote possa estar relacionado. Por exemplo, um *firewall* pode aceitar somente pacotes provenientes da Internet que estejam relacionados a uma conexão iniciada previamente por um *host* interno. *Firewalls* com estado, por considerarem também o contexto de cada pacote, permitem uma política de segurança mais precisa do que os *firewalls* sem estado.

2.3.2 *Firewalls* e a Perda de Transparência da Internet

Os dispositivos de *firewall* em geral, devido à característica inerente de bloquear tráfego entre duas redes, diminuem a transparência da Internet, pois impedem a livre comunicação entre os processos. Este trabalho tenta resolver o problema da transparência nos casos em que um *firewall* é utilizado apenas para proteger a rede interna, e não quando o *firewall* é utilizado também para limitar o acesso à Internet por usuários locais. Por esse motivo, os *firewalls* considerados por este trabalho são os *firewalls* baseados em filtragem de pacotes, configurados com a política de permitir que *hosts* internos iniciem sessões TCP ou UDP com hosts externos, mas impedindo que *hosts* externos iniciem sessões com *hosts* internos. *Firewalls* com políticas mais severas, além de não funcionarem com as técnicas apresentadas neste trabalho, são normalmente utilizados para limitar também o acesso à Internet por usuários locais.

O principal problema dos *firewalls* baseados em filtragem de pacotes, com relação à política de permitir apenas as conexões iniciadas por clientes internos, ocorre quando dois clientes, ambos atrás de seus respectivos *firewalls*, desejam se comunicar diretamente. Nesse caso, mesmo que haja consenso entre os clientes e ambos iniciem a conexão, a comunicação não seria possível, pois o *firewall* de um bloquearia os pacotes SYN de abertura de conexão do outro. Existem, no entanto, diversas técnicas, descritas no próximo capítulo,

que possibilitam o estabelecimento de sessões UDP e TCP entre *hosts* atrás de *firewall* e NAT.

Capítulo 3

Restauração da Transparência da Internet

A restauração da transparência presente na arquitetura original da Internet é uma tarefa não trivial. Se por um lado a eventual adoção do IPv6 em escala global resolveria o problema do esgotamento dos endereços IPv4, dispensando a utilização de NATs, por outro os problemas de segurança – e os *firewalls* – continuariam a existir. Além disso, como um NAT também funciona como um *firewall*, protegendo a rede interna, a migração da Internet para o IPv6 possivelmente causaria a substituição de NATs por *firewalls*, e não a simples eliminação dos NATs como se gostaria de pensar.

Como a restauração verdadeira da transparência original da Internet não parece viável, diversas soluções alternativas têm sido desenvolvidas. As abordagens mais utilizadas, conhecidas como NAT/*firewall traversal* (passagem de NAT/*firewall*), visam o estabelecimento de comunicações com *hosts* atrás de NAT ou *firewall*. Apesar de não funcionar em todos os casos, essas técnicas amenizam a perda de conectividade causada por NATs e *firewalls*.

De maneira geral, dispositivos de NAT e *firewall* impedem que pacotes IP sejam enviados a *hosts* protegidos, à exceção de pacotes identificados como relacionados a uma sessão previamente iniciada por um *host* interno. No caso do TCP, por exemplo, isso implica que *hosts* externos não conseguem iniciar conexões TCP com *hosts* internos. De

maneira similar, um fluxo bidirecional de pacotes UDP só pode ser estabelecido entre um *host* interno e um *host* externo quando o *host* interno envia o primeiro pacote desse fluxo. Além disso, protocolos de transporte menos comuns, como o DCCP (*Datagram Congestion Control Protocol* [27]), geralmente não são suportados por dispositivos de NAT/*firewall* e por isso não podem ser utilizados na comunicação entre um *host* interno e um *host* externo, independente de quem inicia a comunicação.

Devido ao uso predominante do TCP e UDP na Internet, a maioria das técnicas de NAT/*firewall traversal* são feitas especificamente para um desses protocolos. Este capítulo descreve as técnicas mais comuns de NAT/*firewall traversal*.

3.1 Configuração de Dispositivos

Um método para permitir que *hosts* externos estabeleçam sessões TCP ou UDP com *hosts* atrás de NAT ou *firewall* consiste em configurar esses dispositivos manualmente para liberar a passagem de determinados pacotes. No caso de um NAT, por exemplo, pode-se configurá-lo para mapear uma porta externa (TCP ou UDP) a um *endpoint* ⟨endereço IP, porta⟩ interno. No caso de um *firewall*, uma solução é configurar o dispositivo para liberar a entrada de pacotes TCP ou UDP determinados a determinada porta.

Para exemplificar essa abordagem, considere o cliente C de uma aplicação P2P de distribuição de conteúdo. O *host* no qual o cliente é executado tem um endereço IP privado (10.0.0.1) e está atrás de NAT. Suponha que o cliente escuta a porta TCP 1234 por conexões de clientes da Internet. Devido ao NAT, clientes externos não conseguem se conectar ao cliente protegido, pois pacotes de abertura de conexão TCP são descartados pelo NAT, que não tem como determinar o *host* interno ao qual os pacotes devem ser enviados. Para resolver o problema, pode-se configurar o NAT para encaminhar todos os pacotes destinados à porta 1234 para o *endpoint* no qual o cliente C escuta ⟨10.0.0.1, 1234⟩. Dessa forma, pacotes de clientes externos enviados à porta TCP 1234 do NAT não serão descartados mas encaminhados (após tradução) ao *host* interno.

A configuração manual de dispositivos de NAT e *firewall* apresenta diversos problemas. Um deles é a necessidade de se conhecer as portas utilizadas individualmente pelas

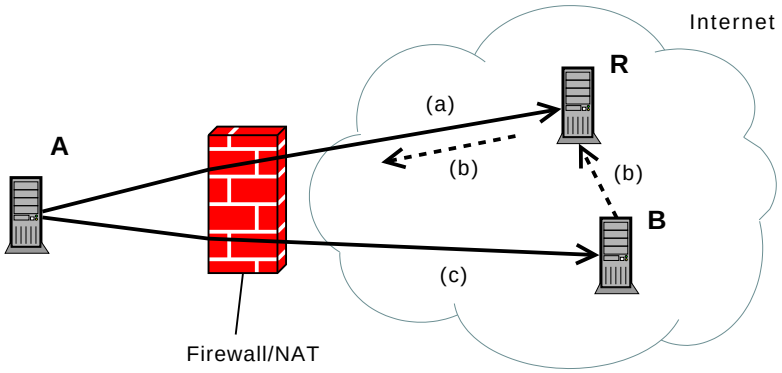

Figura 3.1: *Connection Reversal*. (a) Conexão permanente iniciada por A com o servidor R. (b) Mensagem de B pedindo para A se conectar a B. (c) Sessão TCP ou UDP iniciada por A.

aplicações a serem liberadas. Além disso, o usuário de cada aplicação precisa saber como acessar o dispositivo e alterar suas configurações. Outro problema é que o usuário de uma aplicação pode não ter permissão de acesso ao dispositivo de NAT/*firewall*, o que torna essa solução viável apenas para usuários experientes em redes pequenas ou residenciais.

Na tentativa de resolver os problemas da configuração manual, a arquitetura UPnP (*Universal Plug and Play* [15, 31]) foi desenvolvida. Essa arquitetura é uma coleção de protocolos que visa permitir que as próprias aplicações configurem os dispositivos de NAT/*firewall*. Para que o UPnP possa ser utilizado por uma aplicação, o dispositivo de NAT/*firewall* precisa suportar essa arquitetura. Devido à crescente importância das comunicações P2P na Internet, a maioria dos dispositivos de NAT e *firewall* modernos já possuem suporte ao UPnP.

A principal desvantagem do UPnP é que toda aplicação que necessite da passagem de NAT/*firewall* precisa adicionar suporte a arquitetura, o que dificulta o desenvolvimento de novas aplicações.

3.2 Um Único *Host* atrás de NAT/*firewall*

Um cenário mais simples do problema da perda de conectividade ocorre quando um cliente B, situado em um *host* livre de *firewall* e NAT, deseja iniciar uma sessão TCP ou UDP com um cliente A atrás de NAT ou *firewall*, como mostra a figura 3.1.

Se, por um lado, um *host* da Internet não consegue iniciar sessões com *hosts* atrás

de NAT/*firewall*, por outro o inverso normalmente é possível, isto é, um *host* protegido geralmente não tem problemas em iniciar sessões UDP ou TCP com *hosts* externos livres de NAT/*firewall*. Tendo isso em vista, uma solução comumente usada consiste em inverter o sentido da conexão (*Connection Reversal* [11]): como o cliente B não consegue se conectar ao cliente A, faz-se o contrário, isto é, o cliente A se conecta ao cliente B.

Para implementar essa solução, é necessária a existência de um mecanismo que permita que o cliente B informe ao cliente A sua intenção de realizar uma comunicação. Isso é feito com a ajuda de um servidor bem conhecido R, chamado *rendezvous*, que funciona como um ponto de encontro para os clientes. Antes de tudo, o cliente A deve iniciar e manter permanentemente uma conexão TCP com R. Essa conexão é mostrada pela flecha (a) na figura. Após o estabelecimento dessa conexão, R passa a escutar por pedidos de conexão de clientes que desejam se comunicar com A. Quando um cliente qualquer da Internet, como o cliente B, deseja se comunicar com A, esse cliente (B) começa a escutar em uma porta p. Em seguida, B também se conecta a R e solicita uma conexão com A, informando a porta p (essa solicitação é mostrada pela flecha (b) na figura). O *rendezvous* R, por sua vez, informa ao cliente A que B deseja iniciar uma comunicação, e que está escutando na porta p. Finalmente, o cliente A se conecta diretamente ao cliente B, na porta p (flecha (c)), usando TCP ou UDP.

O servidor R apenas auxilia no processo de estabelecimento de conexão. Após a abertura da conexão entre os clientes, se o cliente B não desejar mais receber conexões de outros clientes, a conexão com R pode ser encerrada. Como a solução apresentada retoma o modelo cliente/servidor, pois B funciona como um servidor para A, essa solução é bastante robusta e pode ser utilizada para se estabelecer tanto sessões UDP quanto TCP.

A seguir são apresentas soluções que permitem que dois *hosts*, ambos atrás de NAT ou *firewall*, se comuniquem.

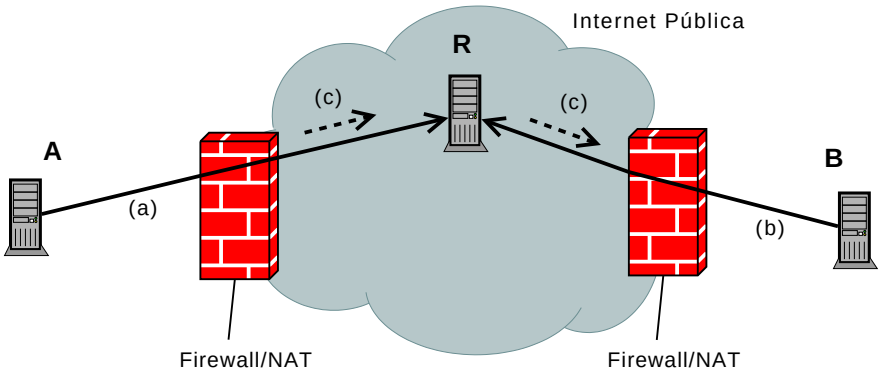

Figura 3.2: *Relaying*. (a) Conexão TCP entre A e R, iniciada por A. (b) Conexão TCP entre B e R, iniciada por B. (c) Mensagem de A para B.

3.3 *Relaying*

Uma solução robusta mas pouco eficiente para a comunicação entre dois clientes atrás de NAT ou *firewall* é o uso de um *relay*, isto é, de um servidor intermediário para repassar todas as mensagens entre os clientes. Por exemplo, como mostra a figura 3.2, dois clientes, A e B, ambos atrás de NAT, desejam se comunicar entre si. Para isso, cada cliente se conecta a um mesmo *rendezvous* conhecido R. Quando um cliente deseja enviar uma mensagem para o outro, esse cliente envia a mensagem para R, que repassa a mensagem para o outro. A mensagem indicada pela flecha (c) na figura mostra um exemplo de uma mensagem enviada do cliente A para o cliente B.

A desvantagem do *relaying* é a necessidade de um servidor que esteja disposto a gastar tempo de processamento e largura de banda para auxiliar na comunicação de outros *hosts*. Além disso, como a comunicação entre os clientes não é direta, essa comunicação não é tão eficiente quanto poderia ser. A principal vantagem dessa abordagem é que, ao retomar o modelo cliente/servidor, que é bem suportado pelos dispositivos de NAT e *firewall*, essa abordagem é bastante robusta.

3.4 UDP *Hole Punching*

O UDP *hole punching* [11] é uma técnica utilizada para se estabelecer uma sessão UDP direta entre dois *hosts*, ambos situados atrás de seus respectivos sistemas de NAT ou *firewall*. A abordagem conta com a ajuda de um servidor conhecido, livre de NAT/*firewall*,

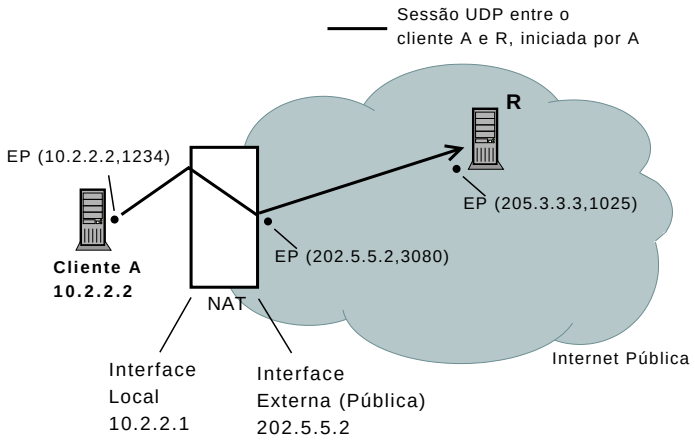

Figura 3.3: Sessão UDP estabelecida entre um cliente atrás de NAT e um servidor externo.

chamado *rendezvous*. O *rendezvous* auxilia apenas durante estabelecimento da sessão e não é utilizado posteriormente. Uma técnica similar é utilizada pelo sistema de VoIP *Skype* [2], para permitir ligações diretas via UDP entre usuários atrás de NAT/*firewall*.

Dois conceitos são importantes para se entender o UDP *hole punching*: *endpoint* e sessão UDP [40, 45]. Um *endpoint* é um par ordenado ⟨endereço IP, porta⟩ que representa um ponto final lógico de uma comunicação. Uma sessão UDP é um fluxo de pacotes identificado por dois *endpoints*. Uma sessão UDP entre um *host* atrás de NAT e um *host* externo, no entanto, possui particularmente 3 *endpoints*. Como ilustra a figura 3.3, considere que o cliente A inicia uma sessão UDP com o servidor R. Os *endpoints* da sessão são os seguintes: o *endpoint* privado de A, isto é, o par ⟨10.2.2.2, 1234⟩ que A acredita estar utilizando para a comunicação; o *endpoint* público de A, isto é, o par ⟨202.5.5.2, 3080⟩ que o NAT utiliza publicamente para a sessão; e o *endpoint* público de R, ou seja, o par ⟨205.3.3.3, 1025⟩ no qual R escuta por pacotes UDP.

O UDP *hole punching* assume NATs do tipo cone. Os NATs do tipo cone são aqueles que associam, para pacotes UDP provenientes de um mesmo *endpoint* privado, sempre a mesma porta pública, independente do *endpoint* de destino do pacote. Ou seja, se um *host* atrás de um NAT do tipo cone envia, utilizando a mesma porta local (privada), um pacote UDP para dois *hosts* externos diferentes, o NAT associa a mesma porta externa (pública) para ambos os pacotes. O UDP *hole punching* em NATs do tipo simétrico é mais complexo, mas pode ser realizado utilizando-se técnicas de predição de portas [19].

3.4.1 STUN

Para permitir que uma aplicação descubra a presença e o tipo de NAT/*firewall* em sua rede, o protocolo STUN (*Simple Traversal of UDP Through NATs*) foi desenvolvido [40]. O STUN é um sistema cliente/servidor UDP simples, baseado em requisição/resposta, que funciona da seguinte maneira. Um servidor STUN é um processo em execução em uma máquina livre de NAT/*firewall* com o objetivo de responder a pacotes UDP enviados por clientes. Um cliente que deseja descobrir a presença de NAT/*firewall* envia uma requisição a um servidor STUN conhecido, o qual responde ao cliente incluindo na resposta o endereço IP e a porta UDP de origem observados no pacote da requisição. Ao receber a resposta, o cliente pode comparar o endereço utilizado na requisição com o endereço contido na resposta do servidor, e dessa forma determinar se houve alguma tradução de endereço e/ou porta. Além disso, outros pacotes de requisição/resposta podem ser utilizados para determinar o tipo de NAT, caso necessário.

Como o STUN permite que *hosts* descubram a presença de NAT/*firewall*, um *host* pode utilizá-lo para determinar se o UDP *hole punching* é realmente necessário. Em outras palavras, se um *host* verificar, por meio do STUN, que possui conectividade livre com a Internet, esse *host* não precisa empregar o UDP *hole punching*. Além disso, um *host* pode também verificar, utilizando o STUN, se o NAT de sua rede é compatível com o UDP *hole punching*. Deve-se ressalvar que, apesar de se chamar *Stunpede*, o sistema proposto neste trabalho não utiliza o protocolo STUN propriamente dito, pois a funcionalidade do STUN está integrada nos *peers rendezvous* do *Stunpede*.

3.4.2 Descrição do UDP *Hole Punching*

Considere A e B, dois clientes atrás de NAT/*firewall* que possuem uma sessão UDP ativa com um *rendezvous* R, como ilustra a figura 3.4. Quando um cliente envia um pacote UDP para R, com o objetivo de se registrar, o *rendezvous* armazena o *endpoint* público do cliente, isto é, o par ⟨endereço IP, porta UDP⟩ observado no cabeçalho UDP do pacote enviado pelo cliente. Suponha que o cliente A deseja estabelecer uma sessão UDP com o cliente B. O UDP *hole punching* funciona da seguinte maneira:

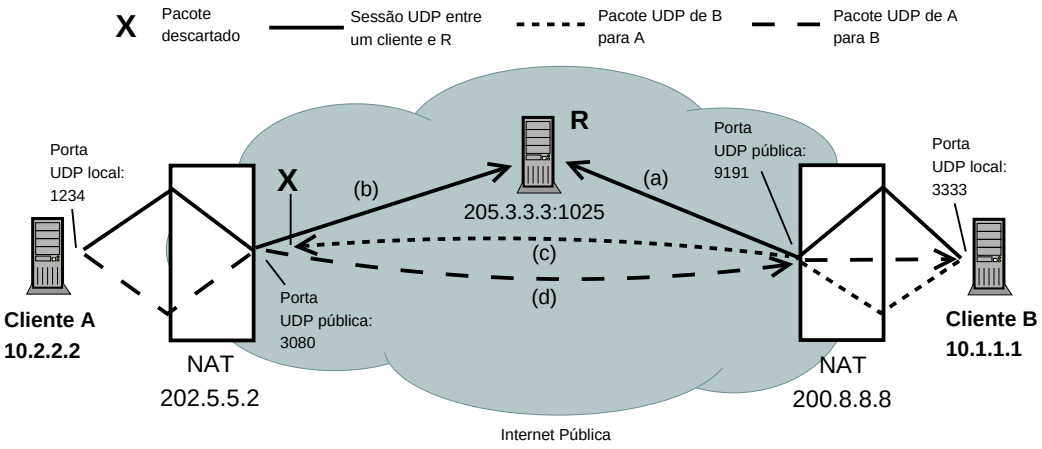

Figura 3.4: Exemplo de *UDP hole punching*. (a) Sessão ativa entre B e o *rendezvous* R. (b) Sessão ativa entre A e o *rendezvous* R. (c) Pacote UDP de B para A, para abrir um furo no NAT de B. (d) Pacote UDP de A para B, para abrir um furo no NAT de A.

- A solicita a R auxílio no estabelecimento de uma sessão UDP com B.
- R envia um pacote a A contendo o *endpoint* público de B, e um pacote a B contendo o *endpoint* de A.
- Ao receberem os pacotes, A e B começam a enviar, de forma intermitente, pacotes UDP para o *endpoint* público do outro, até que uma resposta seja recebida. Cada cliente também escuta por pacotes UDP do outro, enviando uma resposta quando um pacote é recebido. Quando um cliente envia um pacote para o outro, esse pacote abre um "furo" no NAT do cliente que envia o pacote, o qual permite a entrada de pacotes entrantes vindos no sentido contrário.

Um exemplo desse procedimento é ilustrado na figura 3.4. Inicialmente, tanto A como B possuem uma sessão UDP ativa com o *rendezvous* R. O NAT de A associa a porta pública 3080 para a sessão entre A e R, e o NAT de B associa a porta 9191 para a sessão entre B e R. Dessa forma, o *endpoint* público de A é $\langle 202.5.5.2, 3080\rangle$ e o *endpoint* público de B é $\langle 200.8.8.8, 9191\rangle$. O *rendezvous* R verifica o cabeçalho UDP dos pacotes enviados pelos clientes para determinar o *endpoint* público de cada um.

Em seguida, A solicita a R auxílio no estabelecimento de uma sessão UDP com B. R envia o *endpoint* público de A para B, e vice-versa. Ao receber a mensagem de R, B começa a enviar pacotes UDP para o *endpoint* de A, isto é, para $\langle 202.5.5.5, 3080\rangle$. O

primeiro desses pacotes é mostrado pela flecha (c) na figura 3.4. Esse pacote é barrado pelo NAT de A, mas o pacote cria um "furo" no NAT de B que permite a entrada de pacotes enviados por A. Quando A envia um pacote para o *endpoint* de B (flecha d), esse pacote também cria um "furo" no NAT de A que permite a entrada dos pacotes de B. Como o NAT de B já possui um "furo", o pacote (d) consegue passar o NAT e chegar ao cliente B. Finalmente, o cliente B responde a A (não mostrado na figura). Essa resposta passa normalmente pelo NAT de A. Depois desse procedimento, os clientes podem se comunicar normalmente, com cada cliente enviando pacotes para o *endpoint* público do outro. Apesar de no exemplo o cliente B ter enviado um pacote para A antes de A ter enviado o primeiro pacote, isso não é necessário, pois como os clientes enviam pacotes para o outro até o recebimento de uma resposta, a ordem das mensagens no UDP *hole punching* não é importante.

3.4.3 TCP *Hole Punching*

Diversas abordagens [18, 19, 10, 5, 11], conhecidas como TCP *hole punching*, têm o objetivo de estabelecer conexões TCP entre *hosts* atrás de NAT ou *firewall*. O princípio geral dessas abordagens é o mesmo do UDP *hole punching*, ou seja, o envio de pacotes para o outro *host* com a intenção de abrir "furos" no seu próprio NAT, liberando a passagem de pacotes entrantes.

O TCP é um protocolo orientado a conexão que possui um método de estabelecimento de conexão chamado *three-way handshake* [34]. Esse método funciona da seguinte maneira (vide diagrama 3.5). Inicialmente, um servidor S, que deseja receber conexões de um ou mais clientes da Internet, realiza a abertura passiva (*passive open*) de uma porta TCP. A abertura passiva é uma solicitação ao sistema operacional para escutar pedidos de conexão TCP destinados a uma determinada porta. Esse passo não envolve o envio de pacotes pela rede, e consiste apenas de uma comunicação entre a aplicação e o sistema operacional. Em seguida, um cliente C pode estabelecer uma conexão com S, através do *three-way handshake*, que possui 3 passos:

1. O cliente C envia a S um pacote TCP do tipo SYN, solicitando a abertura da

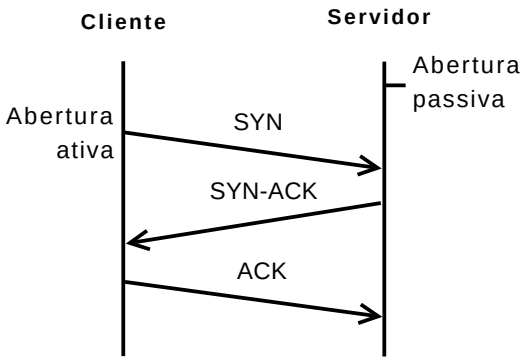

Figura 3.5: *Three-way handshake.*

conexão. Ao enviar esse pacote, diz-se que C realizou uma abertura ativa (*active open*) da conexão.

2. Como resposta, S envia um pacote do tipo SYN-ACK, que representa a aceitação ou reconhecimento da conexão (*acknowledgement*).

3. Finalmente, o cliente C envia um pacote do tipo ACK, confirmando o pacote SYN-ACK recebido.

Após o *handshake*, os processos podem iniciar a comunicação efetiva, enviando pacotes com carga útil (*payload*).

O procedimento necessário para se estabelecer uma comunicação TCP é mais complexo do que o UDP, que não é orientado a conexão, isto é, no UDP os processos apenas enviam pacotes um ao outro, sem necessidade de um acordo (*handshake*). Devido a esse motivo, as diversas técnicas existentes para permitir a comunicação TCP entre *hosts* atrás de NAT/*firewall*, chamadas de TCP *hole punching*, são mais complexas do que o UDP *hole punching*. Além disso, as técnicas de TCP *hole punching* utilizam suposições e comportamentos não padronizados dos dispositivos de NAT/*firewall*, que nem sempre são satisfeitos. Isso faz com que as técnicas de TCP *hole punching* sejam menos robustas que o UDP *hole punching*. Algumas técnicas de TCP *hole punching* são descritas a seguir. Mais detalhes podem ser vistos em [18].

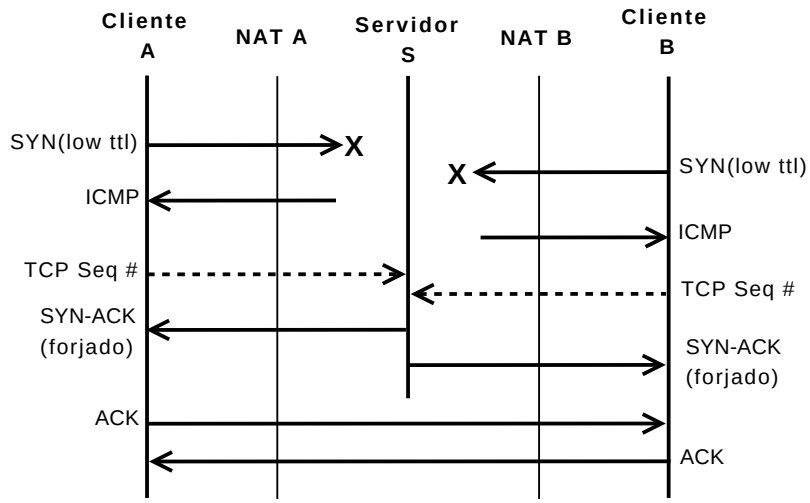

Figura 3.6: Primeira abordagem STUNT.

STUNT

Duas abordagens, chamadas de STUNT, são propostas em [19] para a passagem de NAT/*firewall* com o TCP. A figura 3.6 apresenta o fluxo de pacotes da primeira abordagem STUNT, que funciona da seguinte maneira. Considere A e B, dois clientes atrás de NAT que desejam estabelecer uma conexão TCP entre eles. Inicialmente, cada cliente envia um pacote SYN para o outro, com um TTL (*Time-To-Live*) alto o suficiente para o pacote atravessar o próprio NAT, mas baixo o suficiente para o pacote ser descartado antes de chegar ao NAT do outro cliente. Isso é feito porque um NAT, caso recebesse um pacote SYN não solicitado, poderia responder com um pacote TCP-RST, restartando a conexão e comprometendo o procedimento. Em seguida, cada cliente descobre o número de seqüência TCP utilizado pelo seu sistema operacional para a conexão. Isso é feito inspecionando-se o cabeçalho TCP do pacote SYN, antes do pacote ser enviado. Em seguida, cada cliente informa esse número de seqüência a um servidor conhecido S, e o servidor forja um pacote do tipo SYN-ACK para cada cliente, com o número de seqüência configurado corretamente para que o sistema operacional de cada *host* acredite que o pacote proceda do outro *host*. Finalmente, cada cliente envia um pacote ACK para o outro, finalizando o *three-way handshake*.

Ao terminar o procedimento, ambos os clientes acreditam terem realizado uma conexão TCP ativa com o outro. Os dois principais problemas que a solução enfrenta são os

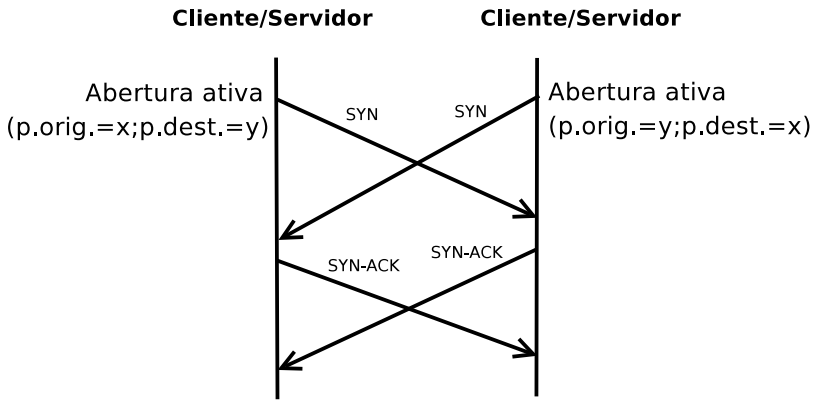

Figura 3.7: TCP *simultaneous open*.

seguintes:

- Como mostra o diagrama 3.6, a mensagem ICMP "TTL excedido" é gerada quando o TTL do pacote SYN enviado pelos clientes atinge o valor zero. Um NAT pode interpretar o pacote ICMP "TTL excedido" como um erro fatal e restartar a conexão, impossibilitando o funcionamento da abordagem.

- É necessária a existência de um servidor que possa forjar um pacote IP. Um pacote forjado pode ser descartado por filtros de segurança, inviabilizando a abordagem.

Na segunda abordagem proposta em [19], apenas um cliente – A, por exemplo – envia um pacote SYN para B, com TTL baixo. Esse pacote cria um "furo" no NAT de A para permitir o recebimento de pacotes de B. Em seguida, o cliente A aborta a conexão e realiza a abertura passiva de outra conexão na mesma porta. Posteriormente, o cliente B inicia uma conexão TCP com o cliente A normalmente, utilizando o "furo" criado por A. Para funcionar, é preciso que o NAT de A aceite um pacote SYN entrante após um pacote SYN ter sido envido para a Internet, uma seqüência de pacotes não vista normalmente. A vantagem da solução é que o servidor não precisa forjar pacotes IP.

Peer-to-Peer NAT

Outra solução de destaque, conhecida como *Peer-to-Peer* NAT, é proposta em [11]. Essa solução utiliza a abertura simultânea de uma conexão TCP (TCP *simultaneous open*

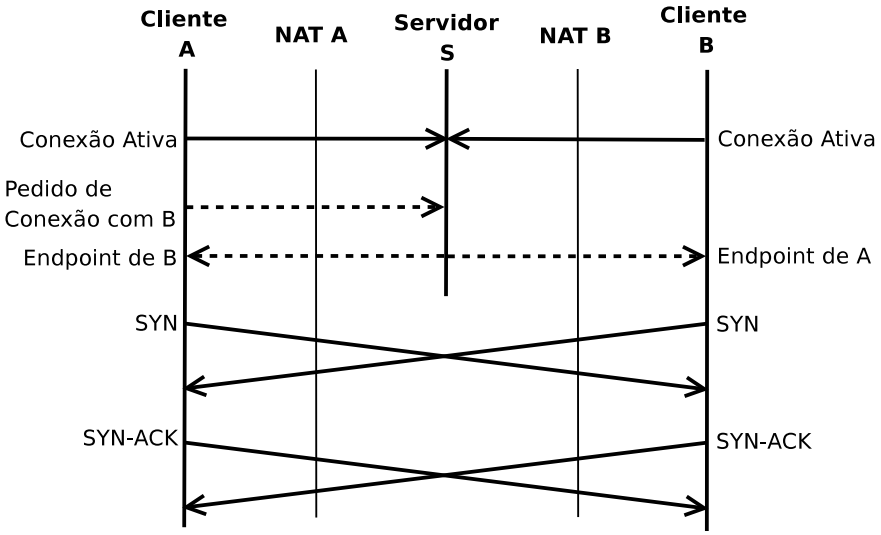

Figura 3.8: *Peer-to-Peer* NAT quando os pacotes SYN se cruzam na Internet.

[34]), um cenário descrito na especificação do TCP que é raramente utilizado. A abertura simultânea (veja figura 3.7) ocorre quando dois *hosts* enviam simultaneamente um pacote SYN para o outro, com as portas de origem e destino invertidas. Ao receber o pacote SYN, cada *host* responde com um pacote SYN-ACK, e a conexão é estabelecida. Apesar de dois *hosts* iniciarem a conexão, obtém-se apenas uma conexão TCP entre eles. Além disso, o *handshake* possui 4 pacotes, ao invés de apenas 3 como ocorre no *three-way handshake*.

Como ilustra a figura 3.8, suponha que os clientes A e B, ambos atrás de NAT, desejam estabelecer uma conexão TCP entre eles. O *Peer-to-Peer* NAT funciona da seguinte maneira. Inicialmente, cada cliente mantém uma conexão TCP ativa com um servidor S. Inspecionando o cabeçalho dos pacotes TCP enviados pelos clientes, o servidor consegue determinar o *endpoint* público de cada um. Em determinado momento, o cliente A envia uma mensagem a S, por meio da conexão TCP ativa, solicitando uma conexão com B. Em seguida, S envia o *endpoint* de A para o cliente B, e vice-versa. Ao receber a mensagem de S, cada cliente inicia uma conexão enviando um pacote SYN ao outro. Se esses pacotes se cruzarem na Internet, como mostra a figura 3.8, ambos os pacotes atravessam os NATs e chegam ao destino. Nesse cenário, cada cliente responde com um pacote SYN-ACK, e a abertura simultânea é completada.

No entanto, como mostra a figura 3.9, se o pacote SYN do cliente A, por exemplo, chegar ao NAT de B antes de B enviar seu próprio pacote SYN, o pacote SYN de A é

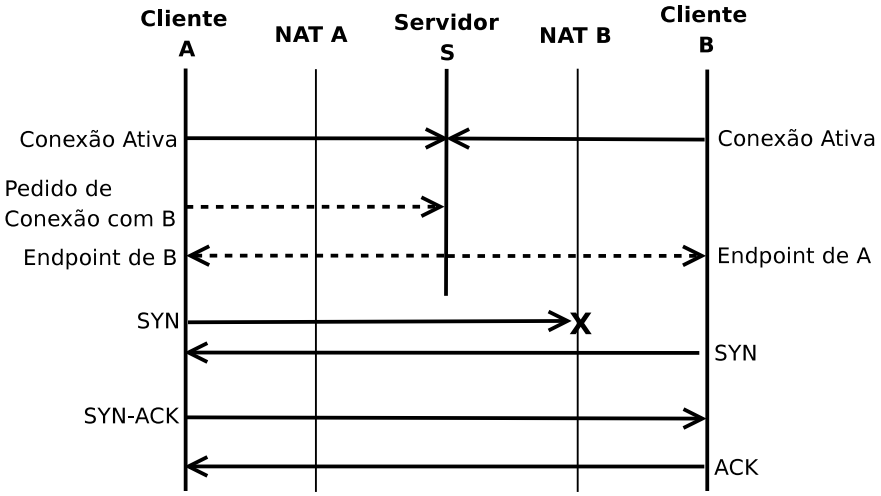

Figura 3.9: *Peer-to-Peer* NAT quando o pacote SYN do cliente A chega ao NAT de B antes de B enviar o pacote SYN.

descartado pelo NAT de B. Em seguida, B envia seu pacote SYN, que chega ao cliente A e gera um pacote SYN-ACK como resposta. Finalmente, o cliente B recebe o SYN-ACK e responde com um pacote ACK, completando o *three-way handshake*. Observe que o cliente A realiza uma abertura simultânea, enquanto o cliente B realiza uma abertura normal de conexão.

A principal vantagem dessa abordagem é que não necessita de forjar pacotes IP. Por outro lado, o *Peer-to-Peer* NAT apresenta os seguintes problemas. Primeiro, é necessário que os sistemas operacionais dos dois *hosts* tenham suporte à abertura simultânea de conexão. Como é um cenário incomum, alguns sistemas operacionais não o implementam. Outro problema é que os NATs dos clientes precisam permitir a entrada de um pacote SYN após um pacote SYN ter sido enviado ao outro cliente, uma seqüência de pacotes não vista normalmente.

Capítulo 4

O *Stunpede*

Este trabalho propõe o *Stunpede*: um sistema P2P que permite que processos executados em *hosts* atrás de NAT/*firewall* se comuniquem de maneira direta e transparente, utilizando qualquer protocolo de transporte.

A idéia central do *Stunpede* é o estabelecimento dinâmico e automático de túneis IPv6-sobre-UDP entre *hosts* atrás de *firewall*/NAT, através da Internet IPv4, para criar a "ilusão" de que os *hosts* estão fisicamente conectados a uma rede IPv6. Quando dois *hosts* desejam se comunicar, uma sessão UDP é estabelecida entre eles, utilizando a técnica UDP *hole punching*. Em seguida, um túnel IPv6 é criado sobre a sessão, permitindo a transmissão de pacotes IPv6 entre os *hosts*. O motivo da escolha do IPv6 se deve ao grande espaço de endereçamento desse protocolo e, em especial, à existência dos endereços locais do tipo ULA (*Unique Local* IPv6 *Unicast Addresses* [22]), utilizados pelo *Stunpede*. No entanto, apesar do *Stunpede* utilizar o IPv6 como protocolo de rede, considera-se que os *hosts* estão fisicamente conectados à Internet IPv4. O restante deste capítulo está organizado da seguinte maneira. A seção 4.1 apresenta uma visão geral do sistema, e a seção 4.2 apresenta estudos de caso e uma avaliação da proposta.

4.1 Visão Geral

O *Stunpede* possui dois tipos de *peers*: clientes e *rendezvous*. Um *peer* cliente é executado em todo *host* atrás de NAT ou *firewall* que deseja se comunicar transparentemente com

outros *hosts*, utilizando o *Stunpede*. O *peer* cliente cria uma interface virtual de tunelamento (IPv6) no *host*, e é responsável por interceptar tráfego destinado a outros *hosts* (*outbound*), assim como injetar tráfego entrante. Os processos com suporte ao IPv6 utilizam a interface virtual normalmente, isto é, como se fosse uma interface real. Quando um processo local envia um pacote IPv6 a essa interface, esse pacote é interceptado pelo *peer* cliente, o qual estabelece uma sessão UDP com o *host* destinatário – por meio de UDP *hole punching*. Em seguida, o pacote IPv6 é encapsulado dentro de um pacote UDP e enviado para o *peer* cliente do outro *host*, através da sessão UDP. Ao receber o pacote, o *peer* cliente destinatário extrai o pacote IPv6 e o injeta em sua própria interface virtual, fazendo com que o pacote pareça ter sido recebido de uma rede física. Esse mecanismo permite a comunicação transparente entre as aplicações.

Já os *peers* do tipo *rendezvous* são *peers* executados em *hosts* sem NAT/*firewall* que auxiliam *peers* clientes a realizarem o UDP *hole punching*. Cada *peer* cliente mantém permanentemente uma sessão UDP ativa com um *rendezvous*. Quando um *peer* cliente A deseja estabelecer uma sessão UDP com um *peer* cliente B, A solicita ao *rendezvous* de B auxílio para a realização de UDP *hole punching* com B. Isso é possível porque cada cliente codifica, dentro de seu endereço IPv6, o endereço IPv4 e a porta UDP do *rendezvous* ao qual está conectado. Dessa forma, a partir do endereço IPv6 de um *peer* cliente (*host*) é possível determinar de imediato o *rendezvous* ao qual o cliente está conectado. A possibilidade de diversos *rendezvous* serem utilizados, em comparação à utilização de um servidor central – implica uma maior escalabilidade e tolerância a falhas, sem aumentar a sobrecarga do sistema. Os *rendezvous* não afetam o desempenho do sistema pois a busca por um *rendezvous* é feita instantaneamente, bastanto se conhecer o endereço IPv6 de um *peer*. Além disso, como os *rendezvous* não se comunicam entre si, eles representam uma maneira simples e eficiente de distribuir a carga do sistema.

Apesar de utilizar o protocolo IPv6 como protocolo de tunelamento, todas as comunicações entre *peers* são realizadas sobre a Internet IPv4. A figura 4.1 apresenta um exemplo do uso do *Stunpede*. Dois *hosts*, A e B, estão atrás de seus respectivos NATs e desejam se comunicar. Um *peer* cliente é executado em cada *host*: o *peer* cliente de A

possui uma sessão UDP ativa com o *rendezvous* W, e o *peer* cliente de B com o *rendezvous* Z. A e B possuem duas interfaces de rede: uma interface física IPv4, com endereços inválidos da faixa 10.0.0.0/8, e uma interface virtual IPv6, criada pelo *peer* cliente.

Suponha que um processo no *host* A, p_A, deseja iniciar uma conexão TCP com um processo no *host* B, p_B. Para isso, p_A solicita ao sistema operacional uma abertura de conexão TCP com p_B (p_A precisa saber tanto a porta TCP em que p_B escuta, quanto o endereço IPv6 do *host* B). O sistema operacional de A então cria um pacote de estabelecimento de conexão TCP (TCP-SYN), e o envia para a interface IPv6. Esse pacote é interceptado pelo *peer* cliente local, o qual verifica que não existe nenhum túnel com o destinatário da mensagem. Por isso, o *peer* cliente precisa solicitar ao *rendezvous* de B (Z) auxílio na realização do UDP *hole punching*.

Para determinar o endereço IPv4 e a porta UDP em que Z escuta, o *peer* cliente extrai essas informações do endereço IPv6 de B. Finalmente, o *peer* cliente envia um pacote UDP a Z (flecha pontilhada), e o processo de UDP *hole punching* é realizado. Após o *hole punching*, a comunicação entre A e B é feita diretamente, sem passar pelos *rendezvous*. A figura 4.2 mostra o cenário após a realização do túnel.

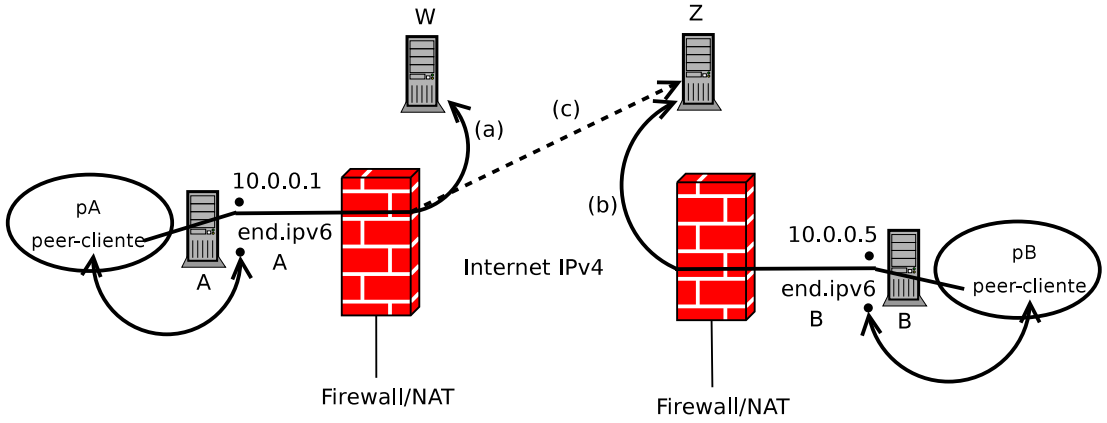

Figura 4.1: Stunpede. (a) Sessão UDP entre A e W. (b) Sessão UDP entre B e Z. (c) Mensagem contendo pedido de UDP *hole punching* entre A e B.

Como o *Stunpede* utiliza o IPv6 nas interfaces virtuais, é necessário que as aplicações possuam suporte a esse protocolo. No entanto, isso não acarreta grandes problemas, pois diversas aplicações e os principais sistemas operacionais já suportam o IPv6. Além disso, na maioria dos casos, a migração de uma aplicação do protocolo IPv4 para o IPv6 é

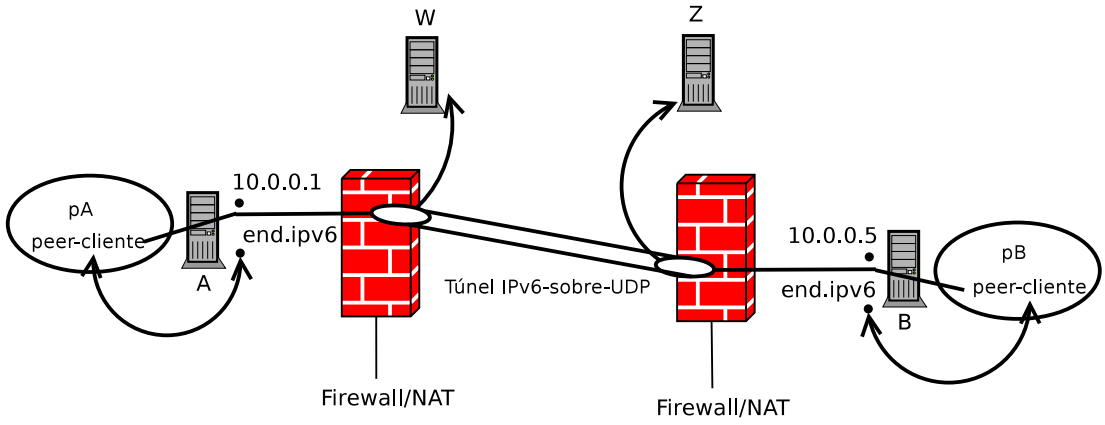

Figura 4.2: Stunpede, após estabelecimento do túnel entre A e B.

bastante simples, uma vez que os protocolos de transporte normalmente usados, UDP e TCP, não mudam com a adoção do IPv6.

O *Stunpede* funciona de maneira similar ao sistema *Hamachi* [42], que também utiliza túneis estabelecidos com *UDP hole punching*. Uma das diferenças é que o *Stunpede* utiliza o IPv6, ao contrário do *Hamachi*, que utiliza o IPv4. O *Stunpede* utiliza endereços locais do tipo ULA, enquanto o *Hamachi* utiliza endereços públicos classe A (5.0.0.0), um bloco de endereços que está atualmente reservado pela IANA (*Internet Assigned Numbers Authority*) e não é utilizado no roteamento global, mas isso não é garantido no futuro. Os endereços ULA utilizados pelo *Stunpede* não são públicos, o que evita conflitos com futuras alocações de endereços, como ocorre com o *Hamachi*. Além disso, o *Hamachi* consiste de um sistema fechado e centralizado, fatores que representam tanto um ponto único de falha como um potencial risco de segurança para os usuários. O *Stunpede* tem como objetivo ser uma solução aberta e distribuída para o problema da conectividade fim-a-fim da Internet. A seguir o funcionamento detalhado dos clientes e *rendezvous* do *Stunpede* é apresentado.

4.1.1 *Peer* Cliente

Um *peer* cliente é um *peer* executado em *hosts* atrás de NAT/*firewall* que desejam participar do *Stunpede*. O funcionamento de um *peer* cliente pode ser divido em duas partes: inicialização (*bootstrapping*) e operação normal, descritos a seguir.

4.1.1.1 Bootstrapping

No primeiro passo da inicialização o *peer* cliente deve estabelecer uma sessão UDP com um *peer rendezvous* qualquer, R, cujo *endpoint* (endereço IP, porta UDP) é conhecido previamente. Para estabelecer a sessão, o *peer* cliente envia um pacote UDP com o objetivo de se logar ao *rendezvous*. O formato desse pacote pode ser visto na figura 4.3. O único campo se refere ao código da mensagem, indicando que se trata de uma mensagem de *log in*.

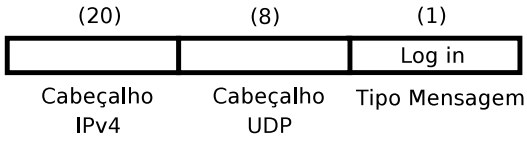

Figura 4.3: Mensagem de *log in*.

O *rendezvous*, ao receber o pedido de *log in*, responde com um pacote UDP indicando se o *log in* foi aceito ou não. Além disso, o *rendezvous* determina o endereço IPv6 que deve ser utilizado pelo *host* cliente. O formato desse pacote pode ser visto na figura 4.4. O primeiro campo indica o tipo da mensagem; o segundo campo contém o IPv6 escolhido pelo *rendezvous* para o cliente; e o campo *status* indica o sucesso da operação.

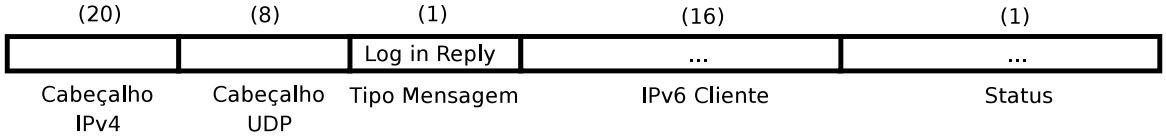

Figura 4.4: Mensagem de resposta ao pedido de *log in*.

O segundo passo do *bootstrapping* é a criação de uma interface virtual e a associação do endereço IPv6 fornecido pelo *rendezvous* a essa interface. Os endereços IPv6 utilizados no *Stunpede* são endereços do tipo ULA (*Unique Local* IPv6 *Unicast Addresses*) [22], cujo formato é apresentado na figura 4.5. Endereços ULA são locais a um domínio e não necessitam de registro junto a autoridades, mas são considerados únicos por toda a Internet. A unicidade ocorre pois cada organização determina o campo *Global ID* aleatoriamente, tornando improvável a existência de repetições. O *Global ID* escolhido para o *Stunpede* tem o valor "63 65 29 4C 47", em hexadecimal, como mostra a figura

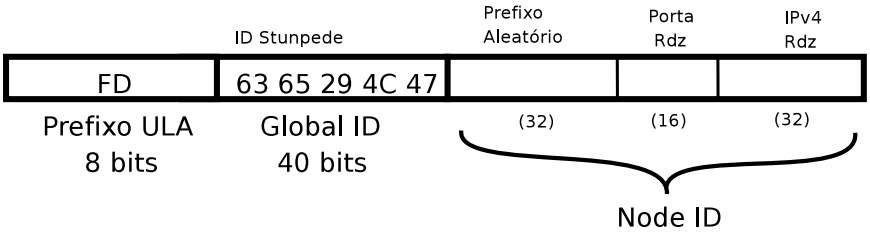

Figura 4.5: Formato do endereço IPv6 de um *peer*.

a seguir. Os primeiros 8 bits dos endereços ULA correspondem ao prefixo IPv6 alocado para endereços desse tipo. O campo *Node ID*, de 80 bits, identifica um *peer* cliente dentro do *Stunpede*, e também permite determinar o endereço IPv4 e a porta UDP do *rendezvous* ao qual o cliente está conectado. Isso é possível pois o endereço IPv4 e a porta UDP do *rendezvous* são codificados dentro do *Node ID*. O prefixo aleatório é um código de 32 bits escolhido pelo *rendezvous* que identifica o cliente de maneira única.

Após criar a interface virtual, o *peer* cliente configura o sistema operacional para interceptar todos os pacotes que são enviados para essa interface. Na implementação de teste do sistema, realizada em plataforma Linux, utilizou-se para isso o *driver* universal TUN/TAP [28]. Além disso, o *peer* também configura as tabelas de roteamento do *host* adequadamente, para que todos os pacotes IPv6 que tenham o prefixo do *Stunpede* ("fd63:6529:4c47::/48"), isto é, destinados a outros *peers*, sejam encaminhados pelo sistema operacional para a interface virtual criada.

4.1.1.2 Operação Normal

Após se logar ao *rendezvous*, o *peer* cliente envia um pacote UDP periodicamente – a cada 10 segundos – para R, com o objetivo de manter a sessão UDP ativa. Esse pacote é do tipo *ping*, e seu formato pode ser visto na figura 4.6. O campo identificador é utilizado para que o *peer* cliente possa combinar a resposta com a pergunta. A resposta do *rendezvous* é uma mensagem do tipo *pong*, a qual tem o mesmo formato da mensagem de *ping*, exceto pelo identificador de tipo de mensagem.

Além de enviar mensagens de *ping* periodicamente para o *rendezvous*, o *peer* cliente também fica esperando por três tipos de mensagens: pacotes IPv6 provenientes da in-

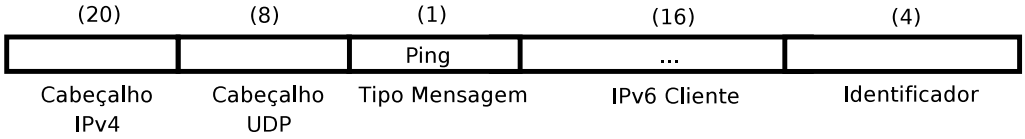

Figura 4.6: Mensagem de *ping*.

terface virtual, pacotes UDP provenientes do *rendezvous*, e pacotes UDP originários de outros *peers* clientes com os quais haja um túnel estabelecido.

Quando um pacote IPv6 é recebido a partir da interface virtual, o *peer* cliente precisa determinar se já possui uma sessão UDP com o *peer* destinatário, por meio da qual possa enviar o pacote. Para isso o *peer* cliente mantém uma tabela com os endereços IPv6 de todos os *peers* com os quais possui uma sessão UDP ativa. Se o *peer* cliente já possui uma sessão UDP ativa com o *peer* destinatário, o *peer* cliente envia o pacote IPv6 dentro de um pacote UDP, como mostra a figura 4.7. Caso não exista uma sessão UDP ativa com o destinatário, é necessário utilizar o método UDP *hole punching* para estabelecê-la, como mostra a próxima subseção.

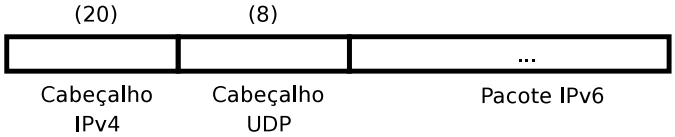

Figura 4.7: Mensagem IPv6 encapsulada em pacote UDP.

Quando o *peer* cliente recebe um pacote IPv6 de outro *peer* cliente, por meio de uma sessão UDP, o *peer* cliente simplesmente "injeta" esse pacote na interface virtual, para que o pacote pareça ter originado de uma rede fisicamente conectada.

4.1.1.3 UDP Hole Punching

Quando um *peer* cliente (A) desejar estabelecer uma sessão UDP com outro cliente (B), cujo endereço IPv6 é conhecido, o procedimento de UDP *hole punching* deve ser efetuado. Para isso, o *peer* cliente A envia um pacote UDP para o *rendezvous* de B, cujo endereço é determinado extraindo a porta UDP e o endereço IPv4 codificados no endereço IPv6 de B. O pacote UDP enviado ao *rendezvous* de B é do tipo UHPReq (*UDP hole punching*

Request), e contém o endereço IPv6 do *peer* cliente local e o IPv6 do *peer* cliente com o qual a sessão UDP precisa ser estabelecida, como mostra a figura 4.8. O procedimento executado pelo *rendezvous* para realizar o UDP *hole punching*, após o recebimento dessa mensagem, é detalhado na próxima subseção.

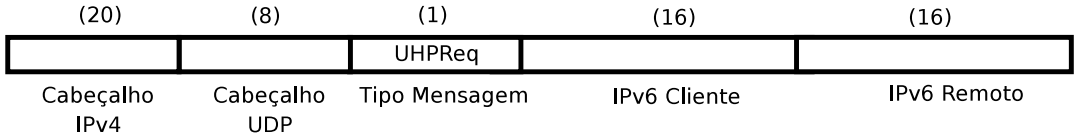

Figura 4.8: Mensagem de *UHPReq* - UDP *hole punching Request*.

4.1.2 *Peer Rendezvous*

Um *peer rendezvous* é um *peer* sem *firewall*/NAT que tem a função de auxiliar *peers* clientes a realizarem o UDP *hole punching*. Um *rendezvous* possui uma tabela com todos os *clientes* logados no momento, isto é, que possuem uma sessão ativa com o *rendezvous*. Cada entrada na tabela se refere a um *peer* cliente e possui os seguintes dados: *endpoint* IPv4 público do cliente, endereço IPv6 do cliente e *timeout*. O campo *timeout* indica o tempo decorrido desde a última mensagem recebida do respectivo cliente, e é utilizado para remover clientes inativos da tabela, isto é, que ultrapassem um determinado *timeout*.

Além de remover clientes inativos de sua tabela, o *rendezvous* também escuta em uma porta UDP por pacotes de *peers* clientes. Existem 4 tipos de pacotes que podem ser recebidos por um *rendezvous*: *log in*, *ping*, *log off* e *UHPReq*, descritos a seguir.

Um pacote do tipo *log in* é utilizado por um *peer* cliente para se registrar junto ao *rendezvous*. Ao receber esse pacote, o *rendezvous* gera um endereço IPv6 para o cliente, conforme visto anteriormente, adiciona o cliente em sua tabela de sessões ativas, e envia um pacote de resposta, informando o endereço IPv6 escolhido.

Um pacote do tipo *ping* é enviado pelo cliente para manter sua sessão ativa. Ao receber esse pacote, o *rendezvous* responde com uma mensagem do tipo *pong*, além de zerar o campo *timeout* do respectivo cliente em sua tabela.

Um pacote de *log off* é utilizado pelo cliente para terminar sua sessão com o *rendezvous*. Ao receber esse pacote, o *rendezvous* remove o cliente da tabela e envia uma mensagem

de confirmação. Se o *rendezvous* não receber nenhum pacote de um cliente por um determinado tempo, o cliente é automaticamente eliminado de sua tabela.

Um pacote de *UHPReq* é enviado por qualquer *peer* cliente a um *rendezvous* com o objetivo de estabelecer uma sessão UDP com um *peer* cliente desse *rendezvous*. Ao contrário das mensagens de *ping* e *log off*, o cliente que envia o pacote de *UHPReq* não precisa estar logado no *rendezvous*, ou seja, ele pode estar logado em outro *rendezvous*. Um pacote de *UHPReq*, como visto na figura 4.8, possui o endereço IPv6 do *peer* cliente solicitante do *hole punching*, assim como o endereço IPv6 do *peer* cliente de destino, isto é, o *peer* com o qual se deseja realizar o *hole punching*. O *rendezvous*, ao receber esse pacote, verifica se o *peer* de destino é um de seus clientes ativos: em caso negativo, o *rendezvous* responde com um pacote de erro ("destino não encontrado"), e em caso afirmativo o *rendezvous* inicia o UDP *hole punching* entre os *peers*. Para isso, o *rendezvous* envia um pacote do tipo *UHPAns* para cada *peer* cliente, informando a um o *endpoint* do outro. O formato desse pacote é mostrado na figura 4.9. Ao receber esse pacote, cada *peer* cliente começa a enviar pacotes para o *endpoint* do outro, de acordo com o método UDP *hole punching*, conforme descrito na seção 3.4.

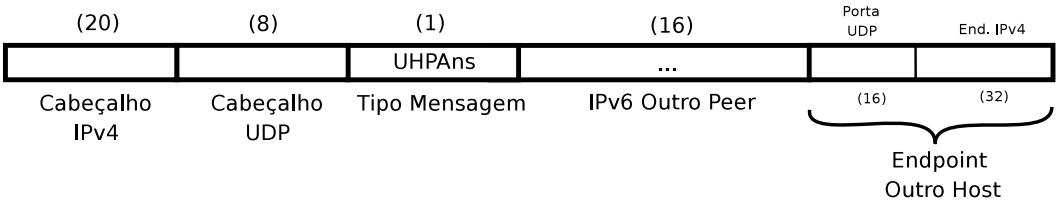

Figura 4.9: Mensagem de *UHPAns* - UDP *hole punching Answer*.

4.2 Estudos de Caso e Avaliação

Esta sessão apresenta um estudo de caso em que se utiliza o *Stunpede* em conjunto com o SNMP (*Simple Network Management Protocol* [21]), com o objetivo de gerenciar recursos distribuídos em múltiplos sistemas autônomos, possivelmente protegidos por *firewall* e NAT. Além disso, também é apresentada uma avaliação da latência adicional causada pelo *Stunpede*, em comparação a uma comunicação convencional, isto é, que utiliza apenas o

TCP/IP.

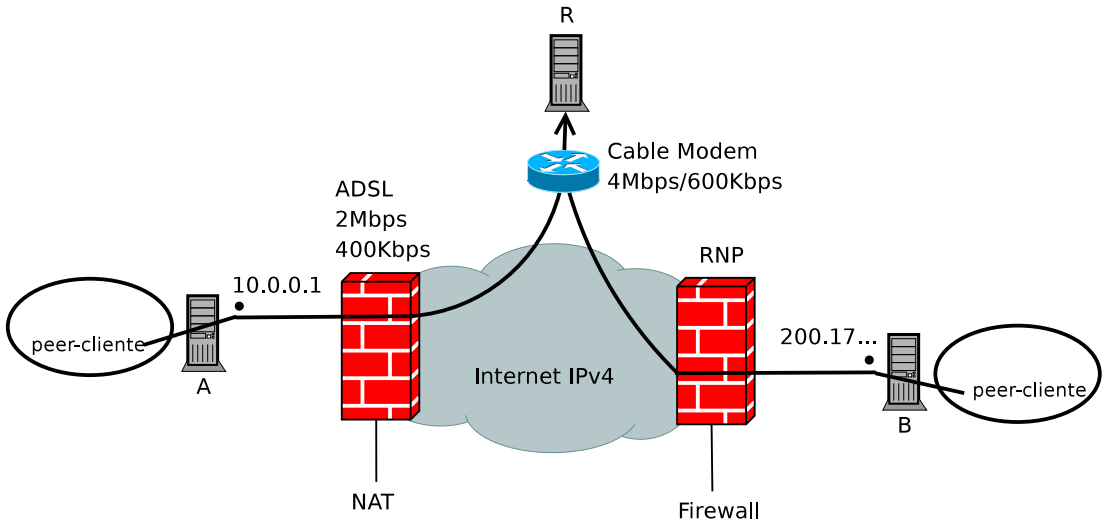

Figura 4.10: Cenário utilizado nos experimentos.

Os *hosts* empregados nos experimentos são ilustrados na figura 4.10 e descritos a seguir. Um *host A* está conectado à Internet através de ADSL, e possui largura de banda de 2Mbps de *download* e 400Kbps de *upload*; um *host B*, situado na rede do Departamento de Informática da UFPR, é conectado à Internet através da RNP, e está protegido por um *firewall* que proíbe conexões TCP e UDP entrantes. Um terceiro *host*, *R*, tem o papel de *rendezvous* e está conectado à Internet por *cable modem*, com largura de banda de 4Mbps de *download* e 600Kbps de *upload*. Inicialmente, antes da realização de cada experimento, um *peer* cliente é executado tanto no *host A* como no *host B*. De forma similar, um *peer rendezvous* é executado em *R*. A título de exemplificação, uma das execuções do *peer* cliente no *host A* causou a criação da seguinte interface de rede virtual, visualizada pelo comando *ifconfig*:

```
user@host-a$ ifconfig

tun0 Encapsulamento do Link: Não Especificado
     endereço inet6: fd63:6529:4c47:240b:17a5:fc8:c915:8888/128 Escopo:Global
     UP POINTOPOINT RUNNING NOARP  MTU:1468  Métrica:1
     pacotes RX:0 erros:0 descartados:0 excesso:0 quadro:0
     Pacotes TX:0 erros:0 descartados:0 excesso:0 portadora:0
     colisões:0 txqueuelen:500
```

```
    RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

Após a execução dos *peers*, é possível utilizar aplicações com suporte ao IPv6 para realizar a comunicação transparente entre *hosts* atrás de NAT/*firewall*. Uma demonstração com o protocolo SNMP é feita a seguir.

4.2.1 SNMP

O SNMP [21] é o arcabouço padrão da Internet para monitorar e controlar dispositivos de rede. No uso típico do SNMP, agentes reportam informações administrativas sobre elementos gerenciados para aplicações de gerência, por meio do protocolo SNMP. No entanto, em diversos casos os agentes e gerenciadores estão em diferentes redes administradas independentemente, o que dificulta a comunicação entre esses componentes, devido a presença de NATs e *firewalls*. Este exemplo demonstra a utilização do *Stunpede* para lidar com esses problemas, permitindo o gerenciamento de dispositivos atrás de NAT e *firewall*.

Para a realização dos experimentos utilizou-se o pacote de software NET-SNMP, o qual implementa o protocolo SNMP para várias plataformas, incluindo o Linux, no qual os testes foram executados. Como o NET-SNMP suporta o IPv6, foi possível utilizá-lo com o *Stunpede* diretamente. Inicialmente, um agente foi instalado no *host A*. Foram necessárias duas alterações na configuração do agente: uma para permitir requisições de *hosts* externos e outra para ativar a escuta na interface IPv6, que não acontece por padrão.

Em seguida, o seguinte comando foi executado no *host B*, para solicitar informações administrativas do *host A*:

```
user@host-b$ snmpwalk 'udp6:[fd63:6529:4c47:5c80:f81a:fc8:c915:8888]' -v1 -c public \
  system

SNMPv2-MIB::sysDescr.0 = STRING: Linux mestre 2.6.17-11-generic \
  #2 SMP Fri May 18 23:39:08 UTC 2007 i686
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (869918) 2:24:59.18
SNMPv2-MIB::sysContact.0 = STRING: jggb@inf.ufpr.br
SNMPv2-MIB::sysName.0 = STRING: mestre
```

```
SNMPv2-MIB::sysLocation.0 = STRING: MainDatacenter
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (1) 0:00:00.01
SNMPv2-MIB::sysORID.1 = OID: IF-MIB::ifMIB
...
SNMPv2-MIB::sysORDescr.1 = STRING: The MIB module to describe \
  generic objects for network interface sub-layers
...
SNMPv2-MIB::sysORUpTime.1 = Timeticks: (0) 0:00:00.00
...
```

O comando *snmpwalk* é utilizado para obter diversos valores de gerenciamento de uma única vez. O endereço IPv6 do *host A* foi passado como parâmetro para o comando *snmpwalk*, entre colchetes, para proteger os caracteres ':' do endereço de outras possíveis interpretações.

4.2.2 Avaliação de Desempenho

Diversos experimentos foram realizados para avaliar o desempenho do *Stunpede*. A configuração de rede e os *hosts* utilizados nos testes foram os mesmos descritos no início da seção 4.2 e ilustrados na figura 4.10. Os experimentos são apresentados a seguir.

Experimento 1 - Latência Inicial

O primeiro experimento tem como objetivo investigar o tempo gasto no procedimento de UDP *hole punching* entre dois *hosts* do *Stunpede*. O experimento consiste em utilizar a ferramenta *ping6* – o comando *ping* com suporte ao IPv6 – no *host A* para "pingar" o *host B* pela primeira vez, isto é, o comando *ping6* é executado logo após a execução dos *peers* clientes para garantir a inexistência de um túnel entre A e B. O tempo de ida e volta (RTT – *Round-Trip Time*) do primeiro *ping* engloba o tempo gasto no processo de UDP *hole punching* entre A e B, assim como o tempo gasto no envio e recebimento de um pacote ICMP (*Internet Control Message Protocol*) pela Internet. O experimento foi executado 200 vezes, e a média do RTT é mostrada na barra da esquerda da figura 4.11. O RTT médio observado foi de 114.227ms, com um desvio padrão de 9.698ms.

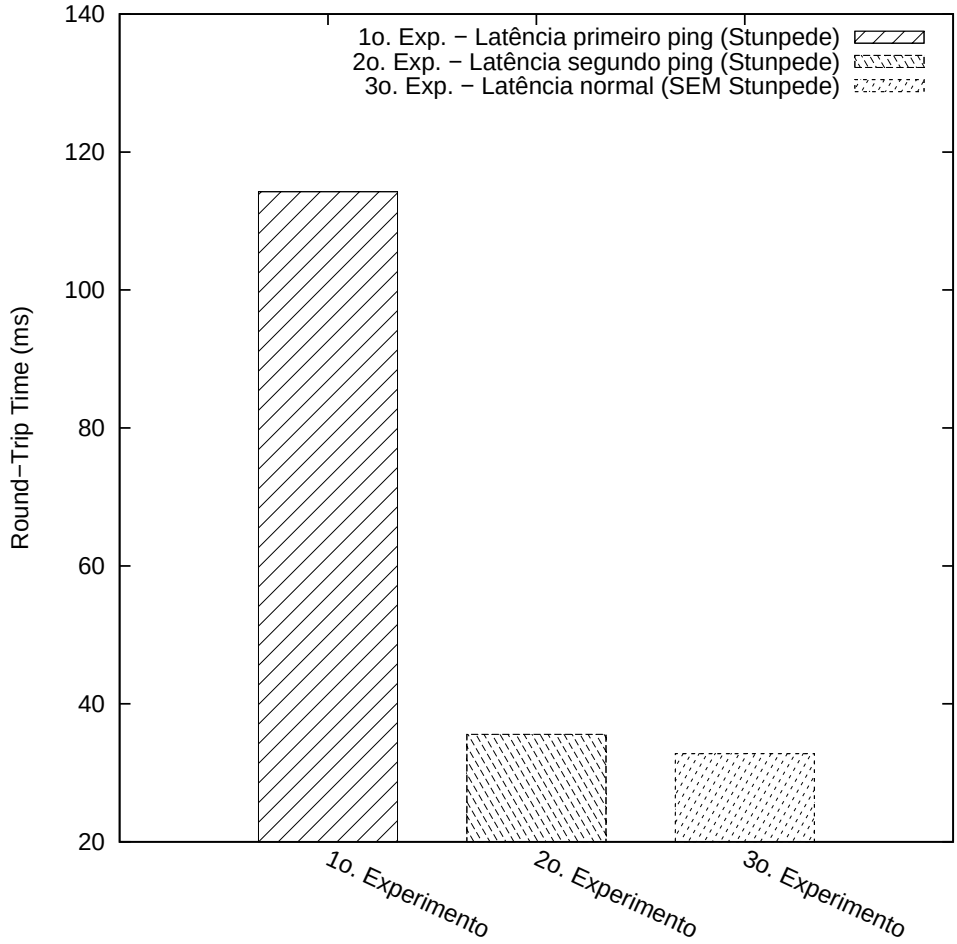

Figura 4.11: Latência média entre os *hosts* com pacotes ICMP de 64 bytes.

Para avaliar a variação da latência em relação ao tamanho do pacote ICMP, o mesmo experimento foi também realizado com pacotes ICMP de tamanhos diferentes do padrão, que é 64 bytes. As médias dessas latências são mostradas na linha superior da figura 4.12. Na média, observou-se um acréscimo de 8.43% na latência quando se aumentou o tamanho do pacote ICMP de 64 para 500 bytes, e de 21.01% quando se alterou o tamanho do pacote de 64 para 1000 bytes.

Experimento 2 - Latência dos Pings Seguintes

O segundo experimento também mede o RTT entre dois *hosts* do *Stunpede*, mas ao contrário do primeiro experimento, nesse caso o comando *ping6* é executado quando já existe uma sessão UDP estabelecida entre os *hosts*. Em outras palavas, o comando *ping6* é executado duas vezes: a primeira para estabelecer um túnel entre os *hosts* e a segunda

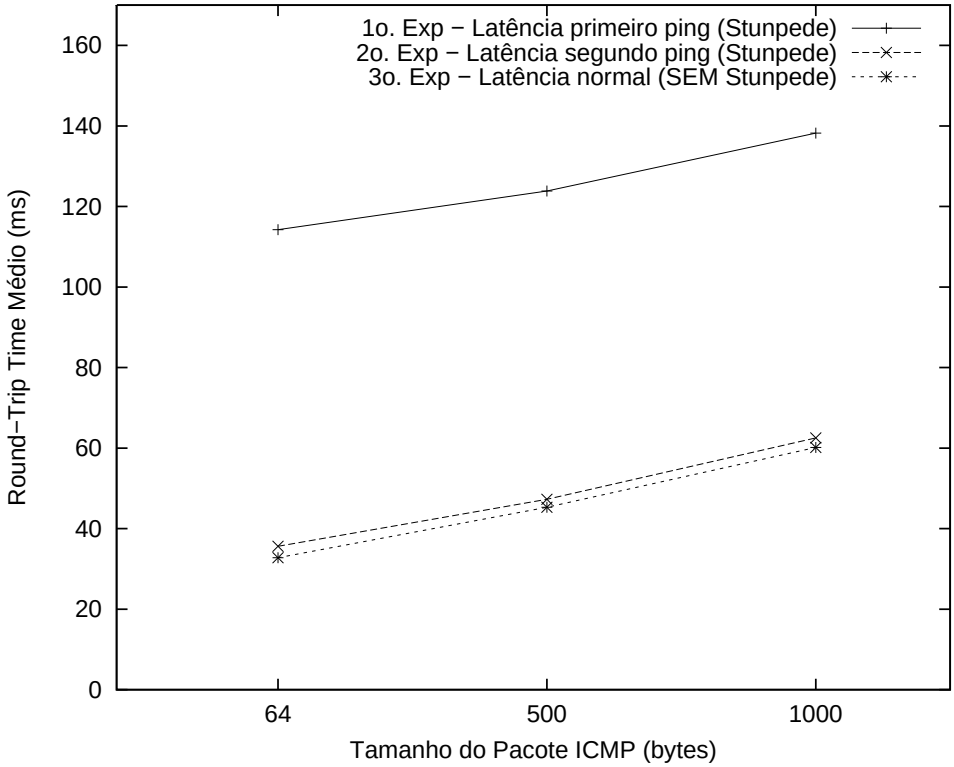

Figura 4.12: Variação da latência entre os *hosts* com o aumento do tamanho do pacote ICMP.

para realizar a medição do RTT. Dessa forma, esse experimento mede apenas o tempo de ida e volta de um pacote ICMP entre dois *hosts* por um túnel já estabelecido, sem incluir o tempo de criação do túnel, como no experimento 1. O experimento foi executado 200 vezes, e a média do RTT é mostrada pela barra do meio da figura 4.11. O RTT médio observado foi de 35.590ms, com um desvio padrão de 2.195ms. Esse experimento também foi realizado com pacotes ICMP de tamanhos diferentes. As médias das latências podem ser vistas na linha do meio na figura 4.12. Na média, observou-se um acréscimo de 32.84% na latência quando se aumentou o tamanho do pacote ICMP de 64 para 500 bytes, e de 75.61% quando se alterou o tamanho do pacote de 64 para 1000 bytes.

Experimento 3 - Latência Sem *Stunpede*

O terceiro experimento mede o RTT entre os mesmos *hosts*, mas sem a utilização do *Stunpede* – isto é, utilizando a ferramenta *ping* tradicional, sobre a Internet IPv4. Como o *firewall* do *host B* não bloqueia pacotes de ICMP, foi possível realizar os testes sem

problemas. A média do RTT para esse experimento, que também foi executado 200 vezes, é mostrada na barra da direita da figura 4.11. O RTT médio observado foi de 32.764ms, com um desvio padrão de 1.419ms. A linha inferior da figura 4.12 mostra o mesmo experimento mas com pacotes ICMP de tamanhos diferentes. Na média, observou-se um acréscimo de 38.27% na latência quando se aumentou o tamanho do pacote ICMP de 64 para 500 bytes, e de 83.62% quando se alterou o tamanho do pacote de 64 para 1000 bytes.

Análise dos Experimentos

Os experimentos apresentados têm como objetivo avaliar dois aspectos do *Stunpede*: a latência para criação de um túnel e a latência adicional causada pelo sistema quando já existe um túnel, em comparação a uma comunicação utilizando o TCP/IP padrão. Em relação ao primeiro aspecto, a latência média do primeiro *ping* entre dois *hosts* utilizando o *Stunpede* (primeiro experimento), considerando pacotes ICMP de 64 bytes, foi 3,486 vezes maior do que a latência média de um *ping* sem o *Stunpede* (terceiro experimento). Como determinou-se experimentalmente que o RTT médio entre A e R é superior ao RTT médio entre A e B, é necessário um intervalo de tempo de pelo menos 3 RTTs para que o *host* A envie o primeiro *ping* para o *host* B (1 intervalo de RTT ou mais para solicitar o UDP *hole punching* ao *rendezvous* e receber a resposta com o *endpoint* de B; 1 intervalo de RTT para enviar um pacote de UDP *hole punching* para B e receber a resposta; e 1 intervalo de RTT para enviar e receber o pacote ICMP propriamente dito). Por esse motivo, a latência de um pouco mais de 3 vezes o RTT entre os *hosts* A e B foi considerada próxima do mínimo teórico esperado para o experimento. Favorece para esse bom desempenho o fato de o *rendezvous* poder ser determinado imediatamente, bastando conhecer o endereço IPv6 do *host* com o qual se deseja comunicar.

Com relação ao segundo aspecto de desempenho, isto é, a latência adicional causada pelo *Stunpede* em relação à uma comunicação normal, observou-se que, considerando pacotes ICMP de 500 bytes, a latência de um *ping* com *Stunpede* (experimento 2) é, na média, apenas 1.97ms (4.35%) maior que um *ping* normal (experimento 3). Outro experimento,

apresentado a seguir, foi realizado para avaliar a largura de banda do *Stunpede*.

Largura de Banda

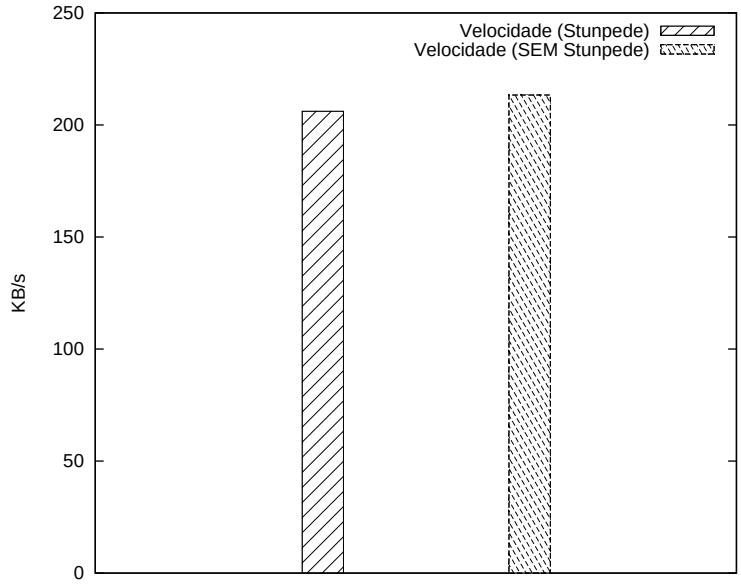

Figura 4.13: Velocidade na transferência de um arquivo de 20MB.

O quarto experimento foi feito com o objetivo de comparar a largura de banda para uma transferência de arquivo em dois cenários: utilizando o *Stunpede* e utilizando o TCP/IP normalmente. Para medir a largura de banda utilizou-se o comando *scp*, o qual tem suporte ao IPv6 e ao IPv4. Um arquivo de 20MB foi transferido entre dois *hosts*, e a velocidade da transmissão foi anotada. Para cada cenário o experimento foi realizado 10 vezes, e a média dessas velocidades pode ser visualizada na figura 4.13. As transmissões utilizando TCP/IP tradicional foram, na média, 3.48% mais rápidas do que utilizando o *Stunpede*. Deve-se observar que um pacote TCP padrão consegue carregar 3.46% mais carga útil que um pacote TCP do *Stunpede*, devido à sobrecarga dos cabeçalho IPv6 e UDP utilizados pelos *Stunpede*. Ou seja, pode-se inferir que o desempenho superior da comunicação TCP/IP ocorre basicamente em função da maior carga útil que é transferida por pacote. Isso sugere que uma maneira de otimizar o *Stunpede* seria a compactação do cabeçalho IPv6, uma vez que nem todos os dados do cabeçalho são utilizados.

Capítulo 5

Conclusão

Este trabalho propôs o *Stunpede*, um sistema P2P que tem como objetivo amenizar a falta de transparência que a Internet enfrenta. O *Stunpede* visa permitir que *hosts* atrás de NAT ou *firewall* se comuniquem de maneira direta e transparente, utilizando qualquer protocolo de transporte. O sistema é baseado no estabelecimento de sessões UDP entre *hosts* atrás de NAT/*firewall*, por meio de UDP *hole punching*, e na criação de túneis IPv6 sobre essas sessões. O *Stunpede* também utiliza *peers* especiais, chamados *rendezvous*, que auxiliam outros *peers* a estabelecerem túneis entre si.

A principal característica do *Stunpede*, em relação a soluções similares detalhadas na literatura, é que a solução proposta não precisa ser implementada individualmente por cada aplicação. Em outras palavras, utilizando o *Stunpede* aplicações com suporte ao IPv6 podem se comunicar com *hosts* atrás de NAT/*firewall* sem serem alteradas. Além disso, a possibilidade de diversos *rendezvous* serem utilizados, em comparação a soluções centralizadas – como o *Hamachi* – implica uma maior escalabilidade e tolerância a falhas, sem aumentar a sobrecarga do sistema.

Uma avaliação de desempenho com o *Stunpede* também foi realizada. Verificou-se que, após o estabelecimento de um túnel entre dois *hosts*, a diferença na latência entre um *ping* "com" e "sem" o *Stunpede* foi bastante pequena. Além disso, o desempenho na transferência de arquivos mostrou-se relacionada à menor carga útil por pacote transferida pelo *Stunpede*.

Uma avaliação do impacto da solução P2P proposta no desempenho do sistema, especialmente comparada com uma solução equivalente baseada em servidor central, fica como trabalho futuro. Outros esforços podem ser feitos com o objetivo de adicionar funcionalidades ao sistema, como a possibilidade de autenticar, criptografar ou compactar o tráfego entre os *hosts*. Além disso, outros trabalhos podem ser feitos para avaliar as implicações relacionadas a segurança causadas pelo *Stunpede*, uma vez que a criação de túneis IP atravessando NATs e *firewalls* potencializa a realização de invasões.

Referências Bibliográficas

[1] R. Atkinson, "IP Authentication Header," RFC 1826, 1995, http://www.ietf.org/rfc/rfc1826.txt [Acessado em 15 de fevereiro de 2008].

[2] S. A. Baset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," Columbia University, New York, NY, Tech. Rep. CUCS-039-04, Sept. 2004.

[3] S. M. Bellovin, "There Be Dragons," in *Proc. UNIX Security Symposium*, 1992, pp. 14–16.

[4] S. M. Bellovin and W. R. Cheswick, "Network Firewalls," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 50–57, 1994.

[5] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig, "NATBLASTER: Establishing TCP Connections between Hosts behind NATs," in *Proc. ACM SIGCOMM ASIA Workshop*, 2005.

[6] B. Carpenter, "Architectural Principles of the Internet," RFC 1958, 1996, http://www.ietf.org/rfc/rfc1958.txt [Acessado em 15 de fevereiro de 2008].

[7] ——, "Internet Transparency," RFC 2775, 2000, http://www.ietf.org/rfc/rfc2775.txt [Acessado em 08 de dezembro de 2007].

[8] B. Cheswick, "An Evening with Berferd in which a Cracker is Lured, Endured, and Studied," in *Proc. Winter USENIX Conference*, 1992.

[9] K. Egevang and P. Francis, "The IP Network Address Translator (NAT)," RFC 1631, 1994, http://www.ietf.org/rfc/rfc1631.txt [Acessado em 08 de dezembro de 2007].

[10] J. L. Eppinger, "TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-ISRI-05-104, Jan 2005.

[11] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-Peer Communication Across Network Address Translators," in *Proc. USENIX Annual Technical Conference*, 2005, pp. 13–13.

[12] P. S. Ford, Y. Rekhter, and H. W. Braun, "Improving the Routing and Addressing of IP," *IEEE Network*, vol. 7, no. 3, pp. 10–15, 1993.

[13] P. Francis, "Is the Internet going NUTSS?" *IEEE Internet Computing*, vol. 7, no. 6, pp. 94–96, 2003.

[14] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy," RFC 1519, 1993, http://www.ietf.org/rfc/rfc1519.txt [Acessado em 15 de fevereiro de 2008].

[15] U. Glasser, Y. Gurevich, and M. Veanes, "High-Level Executable Specification of the Universal Plug and Play Architecture," in *Proc. Annual Hawaii International Conference on System Sciences*, 2002, pp. 3686–3695.

[16] B. Goode, "Voice over Internet Protocol (VoIP)," in *Proc. IEEE*, vol. 90, no. 9, 2002, pp. 1495–1517.

[17] M. B. Greenwald, S. K. Singhal, J. R. Stone, and D. R. Cheriton, "Designing an Academic Firewall: Policy, Practice, and Experience with SURF," in *Proc. Symposium on Network and Distributed System Security*, 1996, pp. 79–92.

[18] S. Guha and P. Francis, "Characterization and Measurement of TCP Traversal through NATs and Firewalls," in *Proc. Internet Measurement Conference*, 2005, pp. 8–18.

[19] S. Guha, Y. Takeda, and P. Francis, "NUTSS: a SIP-based Approach to UDP and TCP Network Connectivity," in *Proc. ACM SIGCOMM Workshop on Future Directions in Network Architecture*, 2004, pp. 43–48.

[20] T. Hain, "A Pragmatic Report on IPv4 Address Space Consumption," *The Internet Protocol Journal*, vol. 8, no. 3, pp. 2–19, 2005.

[21] D. Harrington, R. Presuhn, and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks," RFC 3411, 2002, http://www.ietf.org/rfc/rfc3411.txt [Acessado em 10 de dezembro de 2007].

[22] R. Hinden and B. Haberman, "Unique Local IPv6 Unicast Addresses," RFC 4193, 2005, http://www.ietf.org/rfc/rfc4193.txt [Acessado em 08 de dezembro de 2007].

[23] C. Huitema, "IAB Recommendation for an Intermediate Strategy to Address the Issue of Scaling," RFC 1481, 1993, http://www.ietf.org/rfc/rfc1481.txt [Acessado em 15 de fevereiro de 2008].

[24] K. Ingham and S. Forrest, "A History and Survey of Network Firewalls," University of New Mexico Computer Science Department, Tech. Rep. TR-CS-2002-37, 2002, http://www.cs.unm.edu/~treport/tr/02-12/firewall.pdf [Acessado em 15 de fevereiro de 2008].

[25] J. Kempf and R. Austein, "The Rise of the Middle and the Future of End to End: Reflections on the Evolution of the Internet Architecture," RFC 3724, 2004, http://www.ietf.org/rfc/rfc3724.txt [Acessado em 08 de dezembro de 2007].

[26] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol," RFC 2401, 1998, http://www.ietf.org/rfc/rfc2401.txt [Acessado em 08 de dezembro de 2007].

[27] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion Control Without Reliability," in *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2006, pp. 27–38.

[28] M. Krasnyansky, "Universal TUN/TAP Driver," http://vtun.sourceforge.net/tun/ [Acessado em 08 de dezembro de 2007], 2007.

[29] S. W. Lodin and C. L. Schuba, "Firewalls Fend off Invasions from the Net," *IEEE Spectrum*, vol. 35, no. 2, pp. 26–34, 1998.

[30] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Surveys & Tutorials*, pp. 72–93, 2005.

[31] B. A. Miller, T. Nixon, C. Tai, and M. D. Wood, "Home Networking with Universal Plug and Play," *IEEE Communications Magazine*, vol. 39, no. 12, pp. 104–109, 2001.

[32] A. Muffett, "WAN Hacking with AutoHack: Auditing Security Behind the Firewall," in *Proc. USENIX Unix Security Symposium*, 1995.

[33] L. Phifer, "The Trouble with NAT," *The Internet Protocol Journal*, vol. 3, no. 4, pp. 2–13, 2000.

[34] J. Postel, "Transmission Control Protocol," RFC 793, 1981, http://www.ietf.org/rfc/rfc793.txt [Acessado em 15 de fevereiro de 2008].

[35] ——, "Echo Protocol," RFC 862, 1983, http://www.ietf.org/rfc/rfc862.txt [Acessado em 15 de fevereiro de 2008].

[36] J. Postel *et al.*, "Internet Protocol," RFC 791, 1981, http://www.ietf.org/rfc/rfc791.txt [Acessado em 15 de fevereiro de 2008].

[37] D. Reed, J. Saltzer, and D. Clark, "Active Networking and End-To-End Arguments," *IEEE Network*, vol. 12, no. 3, pp. 66–71, 1998.

[38] Y. Rekhter and T. Li, "An Architecture for IP Address Allocation with CIDR," RFC 1518, 1993, http://www.ietf.org/rfc/rfc1518.txt [Acessado em 15 de fevereiro de 2008].

[39] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets," RFC 1918, 1996, http://www.ietf.org/rfc/rfc1918.txt [Acessado em 15 de fevereiro de 2008].

[40] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489, 2003, http://www.ietf.org/rfc/rfc3489.txt [Acessado em 08 de dezembro de 2007].

[41] J. Saltzer, D. Reed, and D. Clark, "End-To-End Arguments in System Design," *ACM Trans. Comput. Syst.*, vol. 2, pp. 195–206, 1984.

[42] M. Scherer and L. Rev, "Hamachi: The Ultimate VPN?" http://www.cas.mcmaster.ca/~wmfarmer/SE-4C03-06/project/papers/Hamachi.pdf [Acessado em 08 de dezembro de 2007], 2006.

[43] C. L. Schuba and E. H. Spafford, "A Reference Model for Firewall Technology," in *Proc. Annual Computer Security Applications Conference*, 1997.

[44] E. H. Spafford, "The Internet Worm Incident," in *Proc. European Software Engineering Conference*, 1989, pp. 446–468.

[45] P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663, 1999, http://www.ietf.org/rfc/rfc2663.txt [Acessado em 08 de dezembro de 2007].

[46] W. R. Stevens, *TCP/IP Illustrated*. Addison-Wesley, 1994, vol. 1.

[47] C. Stoll, "Stalking the Wily Hacker," *Communications of the ACM*, vol. 31, no. 5, pp. 484–497, 1988.

[48] A. S. Tanenbaum, *Redes de Computadores*, 4th ed. Editora Campus, 2003.

[49] C. Topolcic, "Status of CIDR Deployment in the Internet," RFC 1467, 1993, http://www.ietf.org/rfc/rfc1467.txt [Acessado em 15 de fevereiro de 2008].

[50] J. Wang, "A Survey of Web Caching Schemes for the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 5, pp. 36–46, 1999.