UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA ABORDAGEM COMPLEMENTAR À TRADUÇÃO DE ENDEREÇOS PARA CONECTIVIDADE TRANSPARENTE

LUCAS CLEMENTE VELLA

Uberlândia - Minas Gerais
2012

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS CLEMENTE VELLA

UMA ABORDAGEM COMPLEMENTAR À TRADUÇÃO DE ENDEREÇOS PARA CONECTIVIDADE TRANSPARENTE

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Redes de Computadores.

Orientador:
Prof. Dr. Pedro Frosi Rosa

Coorientador:
Prof. Dr. Lásaro Jonas Camargos

Uberlândia, Minas Gerais
2012

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU

V43a Vella, Lucas Clemente, 1987-
Uma abordagem complementar à tradução de endereços para conectividade transparente / Lucas Clemente Vella. - 2012.
75 f. : il.

Orientador: Pedro Frosi Rosa.
Coorientador: Lásaro Jonas Camargos.

Dissertação (mestrado) ï Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Internet - Teses. 3. Redes de computadores - Teses. I. Rosa, Pedro Frosi. II. Camargos, Lásaro Jonas. III. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Uma Abordagem Complementar à Tradução de Endereços para Conectividade Transparente**" por **Lucas Clemente Vella** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 9 de março de 2012

Orientador: _____
Prof. Dr. Pedro Frosi Rosa
Universidade Federal de Uberlândia

Coorientador: _____
Prof. Dr. Lásaro Jonas Camargos
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Augusto José Venâncio Neto
Universidade Federal do Ceará

_____
Prof. Dr. Renan Gonçalves Cattelan
Universidade Federal de Uberlândia

Dedicatória

Dedico este trabalho ao meu avô, Francisco Clemente,
Para muitos Seu Chico, para os amigos, Chiquinho, para mim, sempre, Vô Chico,
A quem eu fui obrigado a deixar de visitar para que este pudesse ser concluído.

Agradecimentos

Agradeço a Deus, pela chance de existir neste mundo
E a Jesus, por com Ele me conciliar.

Agradeço à minha mãe, Adelina de Barros Clemente
E ao meu pai, Corrado Giovanni Vella
E à minha irmã Miriam Clemente Vella
Por todo amor, carinho, instrução, sustento e apoio
Que eu tive durante toda a minha existência.

Aos meus grandíssimos amigos e colegas
Enrique Fynn, Karina Tânia dos Santos Viana,
Leonardo de Sá Alt e Rodrigo Queiroz Saramago
Parceiros e cúmplices durante essa caminhada acadêmica.

Aos não menos amigos, mas nem tanto colegas
Álvaro Almeida, Bárbara Guerra,
Rafael Teixeira Fernandes e Igor Oliveira de Freitas,
Companheiros nas incontáveis madrugadas longe da cama.

Aos meus outros amigos, que foram os meus mestres
Professores Jamil Salem Barbar, Lásaro Jonas Camargos,
Luís Fernando Faina, Marcelo de Almeida Maia,
Pedro Frosi Rosa, Sandra Aparecida de Amo,
A quem devo o conhecimento que me trouxe até aqui.

Agradeço à UFU, que por tantos anos me acolheu
E também à CAPES, pelo suporte financeiro.

O grande problema com os agradecimentos é que nunca é possível lembrar-se de todas as pessoas e entidades que contribuíram com a obra, nem é possível fazer jus à importância da contribuição de cada uma delas, porque se afinal nos lembramos – e as vezes até esquecemos – de quem mais diretamente nos ajudou, sempre são muito mais os que indiretamente para a obra contribuíram. Afinal, toda a minha história, sucessos e fracassos, risos e choros, culminaram finalmente na feitura deste trabalho como ei-lo. E ainda antes da minha história, toda a história do mundo e da humanidade determinaram a minha existência e a minha realidade. Destes todos, ninguém de quem eu me esqueci ou nunca conheci ou deliberadamente excluí figurará entre os agradecidos. Isto é uma injustiça que eu não posso ter a pretensão de corrigir como uma declaração tão chula como: "Agradeço a tudo e a todos que já existiram." Portanto só resta-me pedir desculpas a todos aqueles que mereceriam aqui figurar, mas tão injustamente foram deixados de fora.

" 'Yes', replied Gutenberg in a serious and dignified tone, 'it is a wine-press in effect, but it is a press from which shortly shall sprout forth floods of the most abundant and the most marvelous liquor that has ever flowed to quench the thirst of man. By it God shall spread his Word; from it shall flow a fountain-head of pure truth. As a new star, it shall dissipate the darkness of ignorance, and cause to shine on men a light hitherto unknown!' "

(Emily Clemens Pearson, "Gutenberg, and the Art of Printing"*)*

Resumo

NAT tradicional é uma técnica extremamente comum, empregada nos roteadores para compartilhar uma única conexão com a Internet entre vários computadores de uma rede local.

Essa técnica foi a responsável pela sobrevivência do IPv4 com sua faixa limitada de endereçamento, e é virtualmente transparente para muitas das aplicações da Internet: aquelas que utilizam a clássica arquitetura cliente-servidor.

Entretanto, aplicações P2P são cada vez mais comuns, e já representam metade do tráfego da Internet no mundo. Essas aplicações necessitam de configurações especiais no roteador para poderem funcionar através do NAT.

Este trabalho apresenta uma abordagem para trazer a transparência original do NAT às aplicações de rede P2P, onde os próprios usuários devem ser acessíveis e prover serviços diretamente para outros usuários.

Nesta abordagem a infraestrutura de rede do sistema operacional é modificada para prover automática e transparentemente a passagem de NAT – necessária a estas aplicações – através das interface padrão POSIX e as chamadas de sistema usuais de acesso à rede, estendendo seus comportamentos padrões.

Nesta nova abordagem, o sistema operacional toma parte no processo do NAT, usando um protocolo da família UPnP para a configuração do roteador e para esconder sua complexidade das aplicações e usuários.

Essas aplicações podem, então, utilizar a interface de rede como se estivessem conectadas diretamente à Internet, não necessitando de implementar nenhum protocolo especial ou de nenhuma ação de configuração extra do usuário, já que o trabalho de passagem de NAT é automaticamente realizado pelo sistema operacional.

Testes utilizando uma implementação de referência e aplicações de rede já existentes suportam a viabilidade da abordagem.

Palavras chave: Tradução de Endereços de Rede, passagem de NAT, UPnP, redes domésticas.

Abstract

Traditional NAT is an extremely common technique, employed on routers in order to share on single Internet connection among many computers of a local network.

This technique has been responsible for the survival of IPv4 with its limited address space, and is virtually transparent to many of the Internet application: those that uses the classic client-server architecture.

However, P2P applications are ever more common, and are already responsible for about half of the world Internet traffic. Those applications require special settings on the router to be functional through NAT.

This work introduces an approach to bring the original NAT transparency to the P2P network applications, where the very peers must be accessible and provide services directly to other peers.

In this approach the operating system's network infrastructure is modified in order to provide automatically and transparently the NAT traversal needed by those applications. This is done through the standard POSIX interface and usual system calls for network access, extending their standard behaviors.

In the new approach, the operating system takes part on the NAT process, using a protocol from UPnP family to configure the router and hides its complexities from applications and users.

These applications then can use the network interface as if they were directly connected to the Internet, not requiring any special protocol implementation or extra user configuration action, since the NAT traversal job is automatically performed by the operating system.

Tests using a reference implementation and existing network applications supports the feasibility of the approach.

Keywords: Network Address Translation, NAT traversal, UPnP, home networks.

Sumário

Lista de Figuras

Lista de Abreviaturas e Siglas

ADSL	*Asymmetric Digital Subscriber Line*
AfriNIC	*African Network Information Center*
API	*Application Programming Interface*
BSD	*Berkeley Software Distribution*
CAD	*Computer Aided Design*
DLL	*Dynamic-link library*
DNS	*Domain Name Service*
FTP	*File Transfer Protocol*
GNU	*GNU's Not Unix!*
HTTP	*Hypertext Transfer Protocol*
HTTPU	UDP HTTP
IEC	*International Engineering Consortium*
IGD	*Internet Gateway Device*
IP	*Internet Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
IPX	*Internetwork Packet Exchange*
ISA	*Industry Standard Architecture*
LAN	*Local Area Network*
MIT	*Massachusetts Institute of Technology*
μTP	*Micro Transport Protocol*
NAPT	*Network Address Port Translation*
NAT	*Network Address Translation*
NAT-PMP	*NAT Port Mapping Protocol*
OSI	*Open Systems Interconnection*
ONU	Organização das Nações Unidas
P2P	*Peer-to-Peer*
PCI	*Peripheral Component Interconnect*
POSIX	*Portable Operating System Interface*
PPA	*Personal Package Archives*

PPP	Protocolo Ponto-a-Ponto
SOAP	*Simple Object Access Protocol*
SPX	*Sequenced Packet Exchange*
SSH	*Secure Shell*
SVR3	*System V Release 3*
TCP	*Transmission Control Protocol*
TLI	*Transport Layer Interface*
UDP	*User Datagram Protocol*
UPnP	*Universal Plug and Play*
UPnP A/V	*UPnP Audio and Video*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*
VFS	*Virtual File System*
VoIP	*Voice over IP*
VPN	*Virtual Private Networks*
WAN	*Wide Area Network*
XML	*Extensible Markup Language*

Capítulo 1

Introdução

Com a ubiquidade da Internet, é fácil se esquecer de todos os desafios técnicos – nos mais variados níveis de engenharia – que foram e ainda são enfrentados para tê-la disponível. Claro que esta não é uma realidade universal, e a Internet está longe de ser um bem comum para a maior parte da humanidade [Argaez 2011]. Mas se é possível esquecer estas dificuldades onde já se tem a Internet funcionando satisfatoriamente, é na sua expansão para além do planejado, ao maior número de pessoas possível, que residem os desafios.

As proporções da Internet já alcançaram níveis muito além do que fora inicialmente projetada [Rekhter et al. 1996]. À medida que crescia, os protocolos e algoritmos originais da Internet foram modificados para se adequar às novas necessidades. Os reflexos dessa adequação podem ser vistos no grande número de *Request for Comments* publicados como padrões da Internet durante sua história [de Souza Pereira et al. 2009].

Dentre estas adequações necessárias ao grande êxito da Internet, está a técnica conhecida como *NAT – Network Address Translation*, ou Tradução de Endereço de Rede. O NAT tornou-se uma técnica comum para resolver um problema enfrentado por muitos usuários da Internet: ter de compartilhar o único ponto de conexão fornecido pelo provedor de Internet entre os vários computadores de sua rede interna.

Já se tornou difícil separar o NAT da Internet contemporânea. Ele integra a infraestrutura de parte significativa das instalações de Internet existentes. Como exemplo, dificilmente se encontra um roteador *wireless* em operação que não seja ele mesmo um tradutor de endereços, ou que não opere abaixo de um, no caso de instalações com mais de um aparelho.

Grande parte do tempo, principalmente ao se utilizar os serviços mais clássicos da Internet, como a *Web*, ou correio eletrônico, não será possível notar se se está trabalhando sob uma rede com NAT ou diretamente na Internet. Isso graças a uma diretiva sob a qual o NAT foi projetado: ele devia ser transparente para o usuário final.

A exemplo do que foi dito anteriormente, o NAT é uma das soluções para um desafio técnico da Internet que, uma vez que se tem em funcionamento, já é possível se esquecer.

E isso é bom. Ao ser projetado para ser invisível, o NAT minimiza o custo de se usar a Internet. Se o usuário já sabia fazer alguma operação na Internet antes do NAT, isso não muda em nada com o NAT. O caso é o mesmo com um programador que já desenvolvia aplicações de Internet antes do NAT: a interface de programação continua a mesma depois do NAT.

Entretanto, esse foi o caso comum. Para certos tipos de aplicação, em geral as que esperam pela iniciativa de um par remoto para funcionar, o NAT não funciona tão bem. A transparência do NAT é garantida somente na arquitetura clássica de cliente/servidor para aplicações de rede, e ainda somente quando o usuário por trás do NAT faz o papel do "cliente".

Para que programas no papel de "servidor" funcionem através do NAT, é necessário realizar configurações e utilizar interfaces além daquelas que são o padrão para a Internet, tornando o NAT uma preocupação para o usuário ou desenvolvedor da aplicação, quebrando sua "invisibilidade".

Isto não era um problema quando este tipo de aplicação estava restrito aos centros de processamento de dados, e arquiteturas de rede alternativas ao clássico cliente/servidor não eram tão comuns entre os usuários finais. Em particular, arquiteturas *peer-to-peer* (P2P) são cada vez mais comuns, e já ocupando mais de 50% da utilização de toda a banda da Internet [Mochalski e Schulze 2009]. Nessas arquiteturas, as aplicações fazem papel de servidores, esbarrando nas limitações de se trabalhar com NAT.

Uma das vantagens originais do NAT, que é sua transparência, vem sendo perdida à medida que as aplicações de usuário final desempenham cada vez mais funções antes restritas aos servidores. Isto obriga usuários e desenvolvedores a se preocuparem com a existência do NAT e suas configurações. O que remete ao escopo deste trabalho: projeto e desenvolvimento de uma solução para a transparência do NAT, que o torne invisível também para aplicações P2P e servidores, trazendo sua transparência original aos novos padrões de uso da Internet.

1.1 Contribuições

Este trabalho traz uma análise do problema da passagem de NAT de uma perspectiva de usabilidade, e das suas implicações para a abstração que a Internet representa. A análise culminará com uma proposta para abordar através do sistema operacional, tendo em vista a transparência dos usuários e aplicações no acesso à Internet.

Será apresentado um método de como adaptar a interface de rede do sistema operacional para lidar automaticamente com a passagem de NAT, e oferecer automaticamente à Internet os serviços de rede das aplicações. A definição do método levará em consideração o modo que as aplicações de rede normalmente utilizam esta interface, bem como questões de controle e segurança oferecidas pela interface.

O trabalho então trará uma implementação de referência do método proposto dentro do núcleo de sistema operacional Linux, provendo uma análise dos mecanismo de comunicação entre modo de usuário e o *kernel* do Linux, mecanismo este necessário à implementação. Esta será utilizada nos testes para validar o método e a abordagem.

1.2 Organização da dissertação

O Capítulo 2 trará uma contextualização da Internet atual e do papel do NAT dentro dela, além de uma análise das tecnologias mais relevantes para passagem de NAT e interface com a pilha de rede.

O Capítulo 3 trará a análise, racionalização e proposta de como obter melhor transparência em uma conexão através do NAT.

O Capítulo 4 descreverá em detalhes a implementação de referência, desde a escolha das tecnologias e decisões arquiteturais até os testes de validação.

Finalmente, no Capítulo 5 serão apresentadas as conclusões e perspectivas para trabalhos futuros.

Capítulo 2

Contextualização

O conceito de "endereçamento" é central para as redes de computadores. Endereçar uma mensagem, neste contexto, significa identificar o destinatário de uma mensagem, provendo os meios para que ela possa ser entregue corretamente.

O NAT é uma importante ferramenta para muitas redes locais, que faz com que os nós na Internet sejam endereçáveis para os nós das redes que empregam o NAT. Entretanto, o contrário não é verdade, e a simples adoção do NAT não torna os nós e aplicações de uma rede local endereçáveis da Internet. Este problema é foco de muitas pesquisas.

2.1 Trabalhos Relacionados

Para tratar o problema de endereçamento de aplicações por trás de NAT nas redes locais domésticas, a abordagem do HomeDNS [Belimpasakis et al. 2007] funciona em nível de resolução de nomes de domínio, com ênfase em serviços HTTP[1], que são comuns em aplicações de *streaming* multimídia. Ele provê uma solução de DNS[2] dinâmico que é capaz de referenciar, a partir da Internet, os vários serviços disponíveis na rede doméstica. Os nomes de domínio do HomeDNS podem, então, ser utilizados para formar as URLs[3] das requisições HTTP. Diferente do presente trabalho, que foca nos problemas de conectividade de aplicações TCP[4] ou UDP[5] já voltadas para a Internet, o HomeDNS foca em estender até a Internet o alcance de serviços HTTP originalmente voltados para a rede local.

Semelhante ao HomeDNS, a solução apresentada por [Haber et al. 2009] provê meios de expor serviços locais da rede doméstica em redes visitantes remotas. O trabalho é especialmente focado em servidores de mídia UPnP A/V[6] e outros serviços voltados para a rede interna, cujo acesso é intencionalmente restrito à rede local e requer um controle

[1] *Hypertext Transfer Protocol*.
[2] *Domain Name Service*.
[3] *Uniform Resource Locators*.
[4] *Transmission Control Protocol*.
[5] *User Datagram Protocol*.
[6] *Universal Plug and Play Audio and Video*.

cuidadoso sobre sua exposição. Isto distingue essa solução da aqui apresentada, que visa os serviços que *deveriam* ser expostos externamente, mas não o são devido à topologia da rede.

Outras soluções se preocupam com as limitações que o uso do NAT tradicional impõe à rede, e propõem soluções. A proposta de [Seah et al. 2009] estende a solução de passagem de NAT já existente que utiliza um *proxy* para intermediar a conexão entre dois pares remotos sujeitos a NAT. Como eles não conseguem se comunicar diretamente, a comunicação é encaminhada através de um *proxy* intermediário. O trabalho sugere ser viável a utilização dos próprios pares na rede da aplicação – aqueles que têm condição para tanto – como *proxy* na comunicação entre pares inalcançáveis entre si. Por um lado, este trabalho aumenta a viabilidade dessa técnica de passagem de NAT, pois alivia o custo de ter de se manter um *proxy* centralizado para encaminhar os dados entre as aplicações, por outro lado, as aplicações ainda deverão ser preparadas para lidar com a passagem de NAT, problema que o trabalho não aborda.

Em *Programable Network Address Translator* (PNAT) [Huang et al. 2010] são levantados alguns problemas comuns enfrentados pelos nós que funcionam através de NAT, dentre elas, a inalcançabilidade dos pares por trás do NAT. Então é sugerida uma nova arquitetura de NAT programável, que resolve todos os problemas levantados. Esta arquitetura, além da possibilidade de executar códigos arbitrários dentro do roteador PNAT, que pode ser usado para resolver certas limitações do NAT, provê um mecanismo de configuração similar ao UPnP, que permite a que os nós internos configurem o *gateway* para a passagem de NAT.

A implantação do PNAT é complexa, exigindo vários serviços auxiliares, como servidor de autenticação e repositório de códigos executáveis no roteador, presentes tanto dentro como fora da rede local. Esta necessidade parece tornar sua utilização em pequenas redes domésticas inviável. Os roteadores e as aplicações de rede devem dar suporte à solução, o que exige uma adaptação tanto dos equipamentos quanto do *software* para seu funcionamento. Assim como o UPnP e outras técnicas de passagem de NAT, o PNAT não vê como problema a necessidade de adaptar as aplicações para poder utilizá-lo. Pelo contrário, cria ainda a necessidade de se programar a tradução de endereços para cada protocolo de aplicação específico utilizado.

Diferentemente desses e de outros trabalhos que provêm alguma solução para que as aplicações possam sobrepujar as limitações do NAT, a proposta descrita neste trabalho provê uma solução para que as aplicações *não tenham* que lidar com as limitações do NAT, pois não é papel da camada de aplicação lidar com este tipo de questão estrutural.

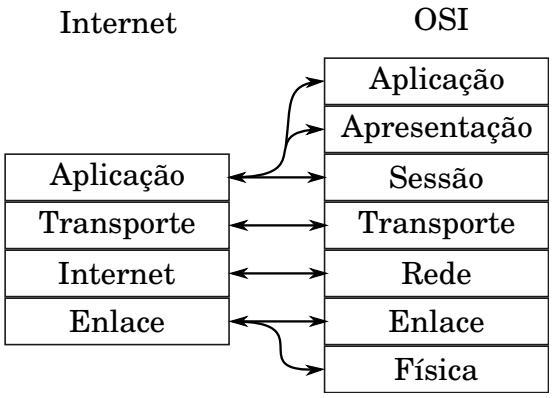


Figura 2.1: Relação entre a arquitetura da Internet e do modelo OSI

2.2 Internet

O modelo de referência *Open Systems Interconnection* – OSI [ISO/IEC 1994] – é uma especificação ISO extremamente influente no projeto das redes de computadores. As redes baseadas no modelo OSI possuem arquitetura em camadas. Cada camada é relativamente independente das outras, e cada uma possui seu próprio mecanismo de endereçamento.

Influenciada pelo modelo OSI, o conjunto de protocolos da Internet – também chamado de TCP/IP – possui quatro camadas [Braden 1989b, Braden 1989a]:

- Camada de Aplicação;
- Camada de Transporte;
- Camada de Internet;
- Camada de Enlace.

Note que o nome da segunda camada (de baixo para cima) é, não coincidentemente, Internet. Na Figura 2.1, que dá uma relação bem aceita entre o modelo OSI e a arquitetura da Internet, é possível verificar que a camada de Internet é a equivalente à camada OSI 3 – camada de rede – que tem como finalidade rotear as mensagens entre diferentes redes locais. Foi justamente desta finalidade de interligar redes locais, desempenhada por esta camada, que se derivou o nome *Internet*.

Internet é um estrangeirismo vindo da língua inglesa; forma contraída do termo *internetworking*, que é o nome dado a um conjunto de redes de computadores interligadas. A camada de Internet desempenha justamente este papel de interligação, e permite que mensagens enviadas por um nó em uma rede sejam entregues a um nó em outra rede geograficamente distante. Tal importância tem este papel para uma rede de proporções mundiais que usamos “Internet” como seu nome próprio.

2.2.1 Endereço IP

A função da camada de Internet é desempenhada pelo protocolo homônimo *Internet Protocol* – IP. Pela relação mostrada na Figura 2.1, o IP é, portanto, colocado como um protocolo da camada 3 no modelo OSI. Ele desempenha bem o papel da camada, sendo responsável por determinar a rota e entregar um datagrama enviado de uma rede local para um outro nó em rede remota.

A versão 4 do IP, identificada com IPv4, embora antiga, é ainda a base da Internet, sendo utilizada em mais de 90% dos computadores nela ligados [Colitti et al. 2010, Google 2012]. Para ser identificado na Internet, cada nó possui um endereço IP, que nesta versão é um número 32 *bits*. Essencialmente, para se usar a Internet, é preciso ter acesso a um nó com um endereço IP válido de onde é possível enviar e receber mensagens.

Um endereço IP válido é um endereço IP público – ou seja, não é uma das faixas de endereço reservadas para uso privado nem de *multicast* – que esteja ligado a um *gateway* (geralmente do provedor de Internet contratado) que possa rotear o tráfego de e para a Internet.

De um ponto de vista socioeconômico, endereços IP válidos são recursos escassos. No Brasil, ao se contratar um serviço de Internet de banda larga via ADSL (caso comum para casas e pequenos escritórios) o contratante recebe somente um IP válido, mesmo no caso frequente em que mais de um dispositivo deva ser ligado à Internet.

Em regiões onde não há disponibilidade de ADSL, é comum o caso de provedores de Internet, muitas vezes ilegais, que possuem somente um IP válido, e dividem seu uso entre seus contratantes.

Por ter 32 *bits* de endereço, existem 2^{32} combinações diferentes de endereços IP possíveis (aproximadamente 4 bilhões). De todas estas combinações possíveis, nem todas podem ser utilizadas como endereço válido, por serem reservadas para fins especiais no protocolo. Esta limitação técnica e a rápida expansão da Internet levaram ao fenômeno conhecido como exaustão dos endereços IPv4, onde todos os endereços disponíveis seriam alocados e não seriam suficientes para suprir a demanda da Internet [Rekhter et al. 1994]. Desde fevereiro de 2011, todas as faixas de endereço IP existentes já estão alocadas [Smith e Lipner 2011], corroborando com a previsão de que os endereços válidos se tornariam cada vez mais escassos.

A exaustão técnica dos endereços poderia não ser tão grave, não fosse a péssima distribuição das faixas de endereços IP. A África possui 54 países soberanos membros da ONU, e sob o controle da AfriNIC – organização que controla a distribuição dos endereços IPs na África – estão 6 faixas de 24 *bits* de endereços IP [IANA 2011], o que permite aproximadamente 100 milhões de endereços para a África inteira, e 1,86 milhões de endereços para cada país, se divididos igualmente. Nos Estados Unidos, o *Massachusetts Institute of Technology* – MIT – possui uma faixa de 24 *bits* própria, tendo à sua disposição

mais 16 milhões de endereços, 9 vezes mais o que um país da África teria direito. É essa má distribuição que torna tão grave o problema da exaustão, limitando o crescimento da Internet nas regiões onde sua expansão for mais tardia.

2.2.2 IPv6

Tendo como metas simplificar o protocolo, removendo funcionalidades que não são mais relevantes no seu contexto de utilização, e resolver o problema da exaustão, a versão 6 do IP foi criada [Deering e Hinden 1998]. Conhecido como IPv6, o protocolo é considerado como sucessor do IPv4 na Internet, mas ainda não conseguiu substituí-lo, sendo mínima a sua adoção.

A principal diferença do IPv6 para o IPv4 é que o seu endereço é de 128 *bits*, o que resulta em um número virtualmente infinito de endereços disponíveis. As 2^{128} combinações possíveis de endereços resolveriam o problema da exaustão, mas não necessariamente aumentaria disponibilidade de endereços válidos para os usuários finais. Essa questão depende não só de fatores técnicos, mas também econômicos ou até mesmo políticos.

Economicamente, o modelo de negócio dos provedores de Internet poderia ser um fator limitante da disponibilidade de endereços válidos, se estes resolverem não disponibilizar uma quantidade razoável de endereços aos seus clientes por uma assinatura de Internet, ou então cobrarem desproporcionalmente mais caro por uma assinatura que disponha de uma faixa de endereços do que cobrariam por uma assinatura com um único endereço, em uma tentativa de diferenciar clientes corporativos de clientes residenciais.

A política da divisão geográfica das faixas de endereço para as regiões e países do mundo também seria um fator determinante na disponibilidade de endereços válidos. O algoritmo de roteamento do IP depende de uma divisão hierárquica dos endereços nas sub-redes. Uma má distribuição inicial poderia comprometer a disponibilidade de faixas de endereços nas pontas das sub-redes, a exemplo do que acontece atualmente com o IPv4: no seu início, faixas de 24 *bits* eram vendidas a organizações que as mantêm até hoje. Atualmente, países inteiros não têm acesso a uma faixa tão grande de endereços, enquanto as faixas adquiridas inicialmente pelas organizações estão extremamente subutilizadas.

É claro que é esperado que situações que limitem a disponibilidade de endereços IPs válidos sejam mitigadas no caso de uma adoção mundial do IPv6, uma vez que já aprendemos a lidar com uma rede de proporções globais se comparado com o que sabíamos quando o IPv4 foi projetado e começou a ser adotado. Os órgãos e políticas para a distribuição das faixas de endereços estão muito melhor desenvolvidos do que eram no início da Internet. Mesmo assim, ainda se trata de uma situação hipotética futura, e não é possível descartar completamente a possibilidade de falta de endereços IPv6 em situações especiais ou localizadas.

2.3 Tradução de Endereço de Rede

O mecanismo de funcionamento do IP é *stateless*, ou seja, não há variação de estado nos nós intermediários da rede durante o seu funcionamento, portanto os roteadores só precisariam de memória equivalente a um autômato finito para serem implementados. Esta decisão de projeto permite que os roteadores sejam simples e extremamente escaláveis. Por outro lado, por não manter o estado da comunicação, cada datagrama em tráfego que chega em um roteador é tratado independentemente dos outros, e precisa conter toda a informação necessária para ser roteado.

Por toda a informação de roteamento estar contida dentro do endereço de 32 *bits*, é de fundamental importância a estrita distribuição hierárquica dos endereços IP dentro de uma sub-rede, seguindo a sua topologia, pois o algoritmo do protocolo utiliza esta organização topológica da rede para encaminhar os datagramas. Essa restrição, e a dificuldade de se conseguir uma faixa de endereços IP contínua, que possa ser organizada em uma sub-rede, torna inviável a criação de redes locais dentro da Internet para a maioria das organizações. A solução encontrada para o problema é criar uma rede privada, que utiliza uma das faixas de endereços IP inválidos reservados para este propósito, e ligar esta rede local privada à Internet utilizando apenas um IP válido, através de *gateway* configurado como tradutor de endereço de rede.

Sob o título genérico de *Network Address Translation* (NAT) se encaixam as várias técnicas de tradução de endereço IP, que servem para estender o mecanismo normal de funcionamento do protocolo de Internet. Essencialmente, qualquer técnica implementada em um *gateway* que burla o funcionamento normal do IP, traduzindo um endereço de uma sub-rede para o de outra, é considerada NAT. Em geral, técnicas de NAT são algoritmos *stateful* (mantêm estados das conexões durante seu funcionamento), portanto sofrem de problemas de escalabilidade inexistente no IP puro. Mesmo assim, são extremamente importantes no cenário atual da Internet, especialmente na criação de redes locais privadas.

O tipo mais comum de NAT, chamado de NAT tradicional após o termo ter sido generalizado [Srisuresh e Egevang 2001], é a técnica utilizada no cenário anteriormente descrito. Nesta configuração, um dispositivo é ligado simultaneamente à Internet e à rede local privada, possuindo dois endereços IP, um válido e um privado. Este dispositivo funciona como o *gateway* padrão da sub-rede.

Em uma rede local IP, todos os endereços pertencem à mesma sub-rede, sendo identificados por uma máscara de *bits*. Os datagramas enviados para endereços IP que não se encaixam nesta máscara, portanto, não deverão ser alcançados dentro da rede física local. O algoritmo então direciona o datagrama ao *gateway* padrão, que como roteador IP, terá meios para encaminhar a mensagem ao destinatário correto.

No caso de uma rede de Internet padrão, o remetente do datagrama possuirá um

endereço válido único na Internet, e o seu *gateway* poderá simplesmente repassar a mensagem ao próximo roteador, que eventualmente o levará ao seu destino. Este não é o caso de uma rede privada, que utiliza sempre uma das 3 faixas de endereço privado reservados para elas, e portanto não são únicas pelo mundo. Por não pertencerem à Internet, os roteadores da Internet descartam os datagramas que utilizam endereços privados.

O processo em um *gateway* com NAT é mais complicado. Ao receber um datagrama endereçado à Internet da rede local, o roteador substitui o endereço privado do remetente pelo seu próprio endereço válido, para então encaminhá-lo para a Internet, como se fosse ele próprio o remetente. Neste processo, o *gateway* registra o remetente real do datagrama e o relaciona com aquela comunicação. Ao receber alguma mensagem da Internet, o *gateway* usa este registro para repassar a mensagem ao seu real destinatário dentro da rede privada, que iniciou a comunicação com nó na Internet.

A operação do NAT tradicional é, portanto, muito mais complexa que a operação de um roteamento IP puro. Sendo ele um algoritmo *stateful* que guarda os estados de todas as comunicações iniciadas pela rede local, o *hardware* que o implementa deve possuir memória e poder computacional para administrá-la. Além disso, deve possuir uma certa inteligência para administrar as conexões estabelecidas, uma vez que o IP, por ser um protocolo *connectionless*[7], não provê nenhum suporte neste sentido. Para tanto, o NAT é obrigado a se utilizar de informações extras além do IP, dos protocolos da camada de transporte, subindo uma camada de abstração na arquitetura da Internet.

Para prover aos nós e aplicações de rede dentro da rede privada as funcionalidades da Internet, o NAT tradicional também opera sobre os dois protocolos de transporte largamente utilizados na Internet: o *User Datagram Protocol* (UDP) [Postel 1980] e o *Transmission Control Protocol* (TCP) [Postel 1981]. Assim como no modelo OSI, os protocolos da camada de transporte da Internet também possuem o seu próprio mecanismo de endereçamento, independente da camada do IP. Tanto o UDP quanto o TCP utilizam o conceito de portas para identificar o seu par remoto, que são números de 16 *bits*.

Ao repassar para a Internet uma mensagem vinda da rede local, o *gateway* deve traduzir também a porta de origem do pacote, que diz respeito a uma porta no nó que originou a mensagem, para uma porta disponível na sua própria pilha de rede, onde o *gateway* deverá receber a resposta daquela mensagem. Uma eventual resposta que chegue pela Internet naquela porta é retransmitida para a porta e IP corretos na rede local.

O NAT tradicional é então dividido em duas partes: o NAT básico, que lida com a tradução dos endereços IP que trafegam através do *gateway*, e o *Network Address Port Translation* – NAPT – que é a operação que cuida da tradução das portas. Ambas as operações feitas de modo transparente para os nós da rede privada, que podem utilizar

[7] O protocolo é dito *connectionless* quando não provê a abstração de conexão contínua entre as partes comunicantes, que deve ser estabelecida, utilizada e depois finalizada.

normalmente a pilha TCP/IP e aplicações de Internet sem que necessitem de nenhuma alteração, tanto para alcançar nós da mesma rede privada quanto da Internet.

Considerando o modo de operação do NAT tradicional, nota-se que, da Internet, não é possível se alcançar diretamente um nó interno à rede privada. Todas as comunicações UDP e conexões TCP devem ser iniciadas de um nó interno, mapeadas pelo NAT, para então ser possível receber mensagens do nó externo. Se uma mensagem chegar ao *gateway* sem que exista uma comunicação prévia na tabela NAT que corresponda àquela mensagem, o *gateway* não terá parâmetros suficientes para determinar quem é o real destinatário da mensagem dentro da rede local. Essa limitação impede o correto funcionamento de aplicações que supõem serem acessíveis por seus pares remotos, sem prévio estabelecimento de comunicação.

2.3.1 Passagem de NAT

Aplicações tipo servidores ou *peer-to-peer* (P2P), em princípio, não funcionariam na Internet se estivessem em uma rede privada com NAT. Afinal, não há como, da Internet, um potencial cliente ou par endereçar o nó da rede interna que executa a aplicação, já que seu endereço é privado.

Para que estas aplicações funcionem corretamente, deve ser empregada alguma técnica de passagem de NAT[8]. Estas técnicas fazem com que o *gateway* repasse para algum nó da rede interna mensagens que cheguem da Internet a uma determinada porta, sem que o nó interno tenha previamente se comunicado com o remetente.

Existem várias técnicas e protocolos que auxiliam na passagem de NAT. Algumas são padronizadas, e exigem suporte oficial dos dispositivos que implementam NAT, como os protocolos *Universal Plug and Play Internet Gateway Device* (UPnP IGD) [UPnP Forum 2010] e o *NAT Port Mapping Protocol* (NAT-PMP) [Cheshire et al. 2008]. Outras técnicas não são padronizadas, e supõem certo funcionamento das implementações NAT e TCP/UDP para funcionarem corretamente, como o UDP e o TCP *hole punching* [Ford et al. 2005].

As aplicações que precisam ser alcançáveis através do NAT devem ser desenvolvidas com este propósito em mente, e implementar alguma técnica de passagem de NAT. Se for uma aplicação que aceita conexões remotas, mas não implementa a passagem de NAT, recai ao usuário ou administrador de rede a tarefa de configurar manualmente o *gateway* para aquela aplicação específica naquele nó da rede interna.

Nenhuma das técnicas disponíveis é universalmente suportada [Muller et al. 2008], e desenvolvedores podem se ver obrigados a implementar mais de uma alternativa. Entretanto, o protocolo UPnP IGD é largamente suportado pelas aplicações de usuário final e roteadores domésticos. Devido à sua ampla disponibilidade, ele será usado no restante deste trabalho.

[8] *NAT traversal*, em inglês.

2.4 *Universal Plug and Play*

O *Universal Plug and Play* (UPnP) é uma iniciativa da indústria para estabelecer um conjunto de protocolos para interconexão de dispositivos domésticos em uma rede TCP/IP. As várias empresas que promovem e o desenvolvem formam o UPnP Forum (`http://upnp.org/`), tendo sido o UPnP inicialmente desenvolvido, no final dos anos de 1990 pela Microsoft, e incorporado, em seu lançamento, no Microsoft Windows ME [Hemel 2006].

O objetivo dos protocolos é facilitar a interação dos usuários com os dispositivos, o que é sugerido pelo nome *Plug and Play*. O nome é uma marca comercial de uma tecnologia que permitia que peças de *hardware* com interface ISA[9] pudessem ser automaticamente configuradas, sem a intervenção do usuário. A marca foi largamente utilizada pela Microsoft no Windows 95. Eventualmente a tecnologia foi superada por interfaces mais modernas, como PCI[10], USB[11] e PCI *Express*, que possuem este mesmo tipo de funcionalidade de autoconfiguração. O nome ainda continuou a ser usado pela Microsoft e originou a marca UPnP, que estenderia as funcionalidades de autoconfiguração até para dispositivos na rede.

Com essa proposta, há vários perfis UPnP, cada um voltado para uma categoria diferente de dispositivos, dentre eles existem:

Audio/Video Trata de dispositivos que armazenam e reproduzem conteúdos multimídia, como aparelhos de TV e *media-centers*;

Home Automation Lida com dispositivos de iluminação e câmeras de segurança;

Networking Gerencia dispositivos da infraestrutura de rede, como *gateways* de Internet e pontos de acesso sem fio;

Printer Descobre e configura impressoras na rede local;

Scanner Descobre e configura aparelhos digitalizadores de imagens;

Telephony Faz interface com aparelhos de telefonia, como dispositivos VoIP[12] e telefones celulares.

Dentre as categorias de dispositivos UPnP, duas são as mais importantes e representam a maior parte de dispositivos UPnP em uso atualmente. A primeira delas é o *Audio/Video*. Abreviado como UPnP A/V, existe uma grande gama de aparelhos domésticos e programas de computador que implementam suas especificações. Em geral é utilizado para se fazer

[9] *Industry Standard Architecture*.
[10] *Peripheral Component Interconnect*.
[11] *Universal Serial Bus*.
[12] *Voice over IP*.

streamming de vídeo e música entre o aparelho que armazena os dados – um computador doméstico – e o aparelho que reproduz – televisores, consoles de *video game*, dentre outros.

A outra categoria de grande relevância do UPnP é a de *Networking*. Mais especificamente, o conjunto de protocolos para fazer a interface com *gateways* de Internet, ou *Internet Gateway Device*, abreviado como UPnP IGD. A principal funcionalidade do protocolo utilizada por aplicativos de rede, em geral P2P e VoIP, é a de configurar o *gateway* para passagem de NAT. Grande parte dos roteadores domésticos implementa o protocolo, que torna o UPnP IGD uma escolha muito viável para se utilizar como passagem de NAT.

Protocolos UPnP são de alto nível, funcionando no nível de aplicação. Funcionam sobre UDP e fazem uso intenso de várias tecnologias de alto nível comuns na *Web*, como HTTP, XML[13] e SOAP[14] [UPnP Forum 2008]. As interfaces oferecidas por um dispositivo UPnP são exportadas como *web services*. Todas as mensagens, requisições e respostas são trocadas através de HTTP, às vezes via uma versão modificada para funcionar via UDP chamada HTTPU. As mensagens são codificadas em XML, como requerido pelo SOAP.

Qualquer operação via UPnP se inicia com o procedimento de descoberta de dispositivos na rede, descrito em detalhes na documentação da arquitetura do UPnP [UPnP Forum 2008]. O procedimento é essencial para a filosofia do UPnP, que implica em nenhuma configuração pelo usuário final, e portanto qualquer dispositivo conectado à rede deve ser automaticamente descoberto e configurado para o uso. O mecanismo utiliza HTTPU sobre o endereço de *multicast* IP 239.255.255.250, que alcança a todos os nós UPnP da rede local [UPnP Forum 2008].

2.4.1 UPnP IGD

Uma vez identificado como IGD, o *gateway* UPnP expõe os seus serviços SOAP aos usuários. Dentre eles, dois são extremamente relevantes na passagem de NAT: os chamados de *AddPortMapping* e *DeletePortMapping*. Eles são usados para adicionar e remover entradas na tabela de redirecionamento. Esta tabela é extra à tabela normal de funcionamento do NAT, que é atualizada automaticamente quando conexões são estabelecidas ou encerradas.

Na tabela de redirecionamento, ficam relacionados todas as portas que oferecem serviços para a Internet através de algum nó da rede local, bem como os protocolos utilizados (TCP ou UDP). Quando chega da Internet um pacote a uma das portas relacionadas, este é direcionado ao nó local correspondente. Em muitos roteadores domésticos esta mesma funcionalidade pode ser configurada manualmente pelo usuário através de uma interface de configuração *Web*.

O comando *AddPortMapping* adiciona uma entrada na tabela de redirecionamento do IGD. Os parâmetros para sua invocação são:

[13] *Extensible Markup Language*.
[14] *Simple Object Access Protocol*.

- *NewRemoteHost*;
- *NewExternalPort*;
- *NewProtocol*;
- *NewInternalPort*;
- *NewInternalClient*;
- *NewEnabled*;
- *NewPortMappingDescription*;
- *NewLeaseDuration*.

O parâmetro *NewRemoteHost* nunca é implementado na prática pelos dispositivos IGD, mas teoricamente poderia ser usado para restringir o mapeamento para somente um nó remoto, de modo que a mensagem seria encaminhada somente caso viesse deste remetente.

Os parâmetros *NewExternalPort* e *NewProtocol* definem o par porta/protocolo que será a chave da entrada na tabela. Cada par único é usado para identificar qual redirecionamento seguir quando um pacote daquele protocolo chega naquela porta.

NewInternalPort designa para qual porta do nó interno o pacote redirecionado será traduzido. Por exemplo, um pacote destinado à porta TCP 80 do *gateway* poderá ser redirecionado à porta 8080 de um servidor *Web* do nó interno *NewInternalClient*. *NewInternalClient* é o endereço IP do nó interno que deve receber as mensagens redirecionadas, formando par com o *NewInternalPort*.

NewEnabled define se o redirecionamento estará ativado ou não. Na prática, é sempre ativado, uma vez que não existem vantagem em manter uma entrada na tabela de redirecionamento desativada. O custo para os clientes de simplesmente adicionar e remover as entradas é o mesmo de ativá-las e desativá-las.

NewPortMappingDescription é uma *string*, com uma descrição relevante para o usuário do que significa aquela associação. Existe somente para facilitar o gerenciamento e não é relevante para o algoritmo. Algumas interfaces de configuração de roteadores exibem esta descrição para o usuário.

NewLeaseDuration especifica por quanto tempo aquela associação ficará ativa. Em geral, as aplicações usam o valor 0 nesse parâmetro, o que equivale a um tempo indeterminado, uma vez que normalmente não sabem de antemão por quanto tempo a associação ficará ativa.

O padrão especifica o significado de vários valores de retorno numéricos para a invocação deste comando. O valor 0 significa que o comando foi executado com sucesso e a associação foi feita ou atualizada. Todos os outros valores especificados representam alguma condição de erro. Dentre eles, existe o valor de retorno 718 (*ConflictInMappingEntry*), que significa que a associação não pôde ser feita porque já existe uma entrada para

outro nó que utiliza o mesmo par protocolo e porta externa que fora requisitado. Como este par forma a chave da tabela, ele deve ser único.

O outro comando importante para a passagem de *NAT* é o *DeletePortMapping*, que, como o nome sugere, serve para remover associações da tabela de redirecionamento. Possui bem menos parâmetros que a *AddPortMapping* pois é necessário somente identificar uma associação que já existe. Os parâmetros são:

- *NewRemoteHost*;
- *NewExternalPort*;
- *NewProtocol*.

O parâmetro *NewRemoteHost*, assim como no comando *AddPortMapping*, não é usado, e os outros 2 formam o par que identifica unicamente a entrada na tabela.

2.4.2 Problemas de Segurança

Segurança de rede e Internet não eram uma preocupação para o mercado de *software* e computadores domésticos no final da década de 1990. Criado neste contexto, o UPnP sofre de graves falhas de segurança na perspectiva atual [Hemel 2006]. Em particular, os protocolos UPnP partem da suposição que os nós da rede local estão sob a mesma autoridade e são confiáveis, o que frequentemente não é verdade. Exemplo comum onde esta suposição falha é uma rede *wireless* pública de algum estabelecimento comercial, onde qualquer indivíduo pode ter acesso.

As especificações do UPnP deixam de exigir muitos cuidados de segurança na implementação, e fica a cargo de cada implementador adicionar suas próprias restrições de segurança. Exemplo dessa falta de cuidado é o campo *NewRemoteHost*, explicado na Seção 2.4.1: não há nenhuma exigência de que o endereço passado como *NewRemoteHost* seja o mesmo endereço do nó que invoca os serviços UPnP IGD. Com isso, é possível a qualquer nó da rede adicionar e remover redirecionamentos para qualquer outro nó da rede.

Muitas implementações permitem esse comportamento potencialmente perigoso, como os roteadores D-Link DI-524 e D-Link DIR-635 RangeBooster N 650. Já outras implementações, como o MiniUPnPd [Bernard 2011] e o Linux-IGD [George et al. 2007] possuem modos de operação que contornam o problema, garantindo que o requisitante possua o mesmo endereço passado na invocação.

A falta de segurança do UPnP é notável ao ponto de os próprios produtos que o implementam fazerem referência ao problema em sua documentação. A implementação de *software* Linux-IGD critica a especificação e os rumos que tomaram o seu desenvolvimento, ao passo que ao final traz uma nota sobre sua tentativa de mitigar o problema [George et al. 2007]. A documentação sobre o UPnP implementado no OpenWrt – um *firmware*

código aberto para modens e roteadores – possui uma seção devotada aos problemas de segurança do protocolo [Frex 2012].

No que concerne ao problema da passagem de NAT, mesmo sendo tão falho e criticado, o UPnP tornou-se a tecnologia dominante no mercado – fato que não é inédito na história da informática. Sua popularidade lhe rendeu ser o meio mais provável de se conseguir a passagem de NAT, sendo, portanto, largamente utilizado pelas aplicações de Internet. Por esse mesmo motivo, o UPnP será utilizado na implementação de referência deste trabalho, sendo fora de escopo solucionar seus problemas de segurança. Dito isto, é perfeitamente possível utilizar alguma outra tecnologia de passagem de NAT mais segura, como o NAT-PMP, para implementar a abordagem aqui proposta.

2.5 Interface POSIX *socket*

Para o desenvolvimento das redes de computadores, tão importante quanto padronizar os protocolo de comunicação entre máquinas, é padronizar o modo de acesso à rede pelos programadores. Se um programa for desenvolvido para usar uma certa pilha de protocolos, e na situação de, por algum motivo, ela precisar ser trocada, não é razoável que o programa tenha de ser reescrito. Nessa situação, um mecanismo padrão de acesso às funcionalidades de rede é desejado, de modo que um programa não esteja profundamente atrelado a alguma tecnologia de rede específica, e possa ser facilmente portado para outra.

A situação é análoga à enfrentada por programas de computador escritos em linguagem de montagem, antes da popularização das linguagens de alto nível, e um programa deveria ser reescrito para funcionar no computador de outro fabricante. Também é análoga à situação enfrentada por programas, já em linguagens de alto nível, mas antes do advento do Unix e da linguagem de programação C, onde a diversidade de sistemas operacionais e compiladores completamente diferentes ainda tornava difícil o reúso do *software*.

Foi justamente um dos rebentos do Unix, o *Berkeley Software Distribution* (BSD), em sua versão 4.2, que trouxe uma revolução para as redes de computadores comparável ao que o próprio Unix havia feito para os sistemas operacionais. Talvez mais importante do que ser a primeira versão do BSD a dispor de uma implementação da pilha TCP/IP – e isso é discutível, afinal o TCP/IP se tornaria o que conhecemos como Internet – foi esta versão que introduziu a abstração de *sockets* e sua API (*Application Programming Interface*).

A biblioteca de *software* da linguagem C conhecida como Berkeley *Sockets* possuía as chamadas `socket()`, `connect()`, `bind()`, `listen()`, `accept()`, `send()`, `recv()`, além de outras que viriam a formar o padrão *Portable Operating System Interface* (POSIX) *Sockets*.

Sockets foram desenvolvidos inicialmente como a interface para a pilha TCP/IP do BSD. Em contrapartida a ela, foi desenvolvida para o Unix *System V Release 3* (SVR3) a interface *Transport Layer Interface* (TLI), que seria independente de protocolos e em conformidade com o modelo OSI. O SVR3 também possuía a interface de *sockets*, incorporada do BSD,

mas com a expectativa de que o modelo OSI eventualmente substituísse o TCP/IP, o uso do TLI no desenvolvimento das aplicações era incentivado.

2.5.1 O padrão POSIX

POSIX é um padrão da indústria de *software* que tem por objetivo uniformizar a interface dos sistema operacionais com os programas, e ajudar a mitigar o ainda existente problema da portabilidade do *software*. Como derivado da família de sistemas operacionais Unix, assim como ele, as interfaces de programação POSIX são dadas na linguagem C.

A última versão do POSIX foi publicada em 2008, e ele vem sendo bem sucedido em sua tarefa de padronizar as interfaces dos sistemas operacionais: grande parte dos sistemas operacionais utilizados na atualidade são, se não totalmente, bastante compatíveis com o padrão POSIX. Graças a ele, aplicações escritas para Linux, MacOS X e FreeBSD – citando três sistemas operacionais comuns – são facilmente portadas entre eles, muitas vezes sem ser necessário alterar uma só linha de código.

Um sistema que notavelmente ainda falha em seguir o POSIX, ao menos em suas versões mais comuns, é o Windows. Isto combinado à sua altíssima penetração no segmento de computadores domésticos e escritórios, faz com que o problema da portabilidade de *software* entre sistemas diferentes seja grave e atual. Em seu estado atual, o problema basicamente implica que um programa pode ser desenvolvido ou para POSIX, ou para Windows.

Mesmo não seguindo o POSIX, a API de rede para Windows foi baseada no Berkeley *Sockets*. Chamada de Winsock, apresenta grande semelhança com o POSIX *Sockets*, e se não é totalmente compatível, utiliza as mesmas abstrações e nomes de funções, o que torna não muito difícil portar programas entre elas.

2.5.2 Interface Genérica

Mesmo tendo sido feito inicialmente como acesso ao TCP/IP, não há nada no projeto da API de *sockets* que a impeça de ser usada para outras pilhas de protocolos. Pelo contrário, na criação de um *socket* é possível determinar qual o seu domínio, ou seja, qual família de protocolos de comunicação será usada. A família mais comum para aplicações em rede, é claro, é a da Internet, `AF_INET`, mas não somente para comunicação em rede a interface é utilizada, podendo-se, por exemplo, utilizar o domínio `AF_UNIX` (conhecido como *sockets* Unix) para se obter um mecanismo de comunicação entre processos locais.

Quando se sobressaíram definitivamente sobre o TLI, *sockets* se tornaram, além de um padrão POSIX, um padrão de fato da indústria para acesso à comunicação em rede. Sua abstração básica, chamada de *socket*, representa o canal de comunicação. Um processo que o utiliza possui ao menos uma das pontas de um *socket*, por onde as informações são enviadas e recebidas, tendo por identificador uma variável do tipo `int` da linguagem C.

Além da Internet e da comunicação local Unix, nos sistemas operacionais modernos o *socket* é utilizado para se comunicar através de qualquer protocolo de rede que o sistema operacional implemente, como IPX/SPX[15] (`AF_IPX`), X.25 (`AF_X25`) – ambos presentes no Linux – ou a nova versão da Internet com `AF_INET6`, comumente presente nos sistemas operacionais atuais.

Também há a possibilidade, no momento da criação do *socket*, de selecionar o tipo da comunicação desejada, podendo ser (em nível decrescente de abstração):

`SOCK_SEQPACKET` Provê um canal de comunicação confiável, orientado a mensagens. É garantido que mensagens enviadas serão entregues em ordem, e com os seus limites preservados, de modo que o receptor saiba onde termina uma mensagem e começa a próxima.

`SOCK_STREAM` Oferece um canal orientado a fluxo de dados. Tudo que for enviado é garantidamente entregue, em ordem, e sem duplicidade ao destinatário, mas não existe a noção de mensagens. Não há garantias de que os dados enviados não foram separados ou aglomerados durante o transporte, portanto o receptor não tem como distinguir as unidades dos pacotes de informação como foram enviados. No TCP/IP este serviço é implementado pelo protocolo de transporte TCP.

`SOCK_RDM` Com a sigla de *reliable datagram*[16], este tipo de comunicação garante a entrega das unidades de mensagem de mais baixo nível, os datagramas, mas não garante sua ordenação nem impede que haja duplicidade de entrega, que pode ocorrer durante o roteamento.

`SOCK_DGRAM` O serviço oferecido aqui é bem mais simples que os anteriores, não possuindo as abstrações e garantias dos anteriores. Neste tipo de serviço, a comunicação pode ser quase tão rápida quanto as camadas inferiores da rede o permitem, com a desvantagem de estar sujeita às mesmas limitações e inconsistências pertinentes a estas camadas. Em particular, se o protocolo de rede não garantir a entrega do datagrama, e for passível de embaralhá-los ou duplicá-los, o usuário estará sujeito a tudo isso. Todos estes problemas estão presentes no IP, e portanto, também no UDP, que é o protocolo que implementa este tipo de *socket* na Internet.

`SOCK_RAW` Dá a possibilidade de ignorar as camadas da pilha de rede, sendo assim uma ferramenta de baixo nível. Quando utilizado, faz com que o *socket* ignore a pilha de rede implementada em *software*, permitindo alcançar diretamente alguma das camadas inferiores. Este tipo de *socket* se desvia muito da categoria formada pelos tipos anteriores, no sentido de que os outros provêm um certo tipo de comunicação em uma dada família de protocolos, enquanto este permite simplesmente ignorar

[15] *Internetwork Packet Exchange*/*Sequenced Packet Exchange*.
[16] Datagrama confiável.

a família de protocolos. Para este uso, `SOCK_RAW` é desencorajado no Linux sendo preferível usar a família de protocolos especial `AF_PACKET`.

Nem todos os tipos de *sockets* são implementados em todas as famílias de protocolos. O IP por exemplo implementa somente os tipos `SOCK_STREAM`, `SOCK_DGRAM` e `SOCK_RAW`. Os nomes dos tipos têm significância semântica, e portanto não estão diretamente atrelados a nenhum protocolo de transporte, mas sim às abstrações (e garantias) que os tipos de comunicação devem oferecer. Deste modo, cabe às famílias de protocolos que implementam a interface *socket* definir qual protocolo estará disponível em cada um dos tipos, conforme as garantias dadas.

2.5.3 Endereçamento

Para estabelecer uma comunicação, é necessário haver algum mecanismo de endereçamento. Na interface POSIX *Sockets*, cada ponto de comunicação pode ser associado a um endereço, que permite ao par remoto identificar aquele *socket*, enviar-lhe mensagem ou estabelecer uma conexão, dependendo do tipo do *socket*.

Por sua natureza genérica, o endereço de *socket* é dado também por um tipo genérico de dados, que possui uma implementação específica em cada família de protocolos. Como a tipagem de dados da linguagem C é estática, um registro que represente o endereço contém, em seu inicio, o identificador da família de protocolos ao qual ele se refere, de modo que possa ser interpretado dinamicamente.

A todas as chamadas da interface que recebem algum endereço, é passado um ponteiro para a estrutura em que os dados de endereçamento estão armazenados, seguido do tamanho daquela estrutura. Sem essa informação, a função não conseguiria determinar o tamanho real dos dados recebidos, já que ela ignora a tipagem estática da linguagem para interpretar dinamicamente seu conteúdo. Por esse mesmo motivo, o tipo de dados que representa um endereço de alguma família de protocolos, por exemplo o `struct sockaddr_in` do TCP/IP, é diferente do tipo genérico utilizado na interface, o `struct sockaddr`, para o qual o tipo original deve ser coagido ao ser passado para a função.

No caso do TCP/IP, seu endereço `struct sockaddr_in` é um registro com três campos. O primeiro, como os endereços de qualquer família de protocolos, especifica a qual família pertence aquele endereço, devendo sempre conter o valor `AF_INET`. O segundo campo contém a porta, que é um valor de 2 *bytes big-endian* com o endereço do protocolo de transporte. Finalmente, o último campo contém, também em *big-endian*, os 4 *bytes* do endereço IP, da camada de Internet.

As três principais funções que utilizam um endereço são `sendto()`, `connect()` e `bind()`. A primeira é utilizada nos tipos de comunicação que não são orientados a conexão, como no UDP, e serve para enviar diretamente uma mensagem a um par remoto. Já a função `connect()` é principalmente para os *sockets* orientados a conexão, como os que

utilizam o TCP. Esta função estabelece a conexão com o par remoto indicado no endereço, e a partir daí o que for enviado e recebido através deste *socket* será, de/para o par conectado.

Enquanto as duas primeiras funções são utilizadas pelo lado ativo da comunicação: quem envia a mensagem ou quem inicia a conexão, a outra é utilizada principalmente no lado oposto, o passivo. A função `bind()` serve somente para associar um endereço a um *socket*, deixando-o apto a receber mensagens, se não orientado a conexões. Quando orientado a conexões, o endereço associado determina quais os recursos locais da rede serão utilizados por aquele *socket*, podendo então ser marcado, com a chamada `listen()`, como um *socket* passivo apto a receber conexões.

Capítulo 3

Abordagem para Abstração do NAT

O IP foi projetado para interconectar redes remotas, possivelmente a longas distâncias. Então, qual a grande vantagem de se utilizar IPs privados e a pilha TCP/IP para se comunicar localmente em uma rede de computadores? Na era da Internet é fácil responder essa pergunta, pois ao se utilizar TCP/IP localmente, qualquer aplicação de rede feita para a Internet funcionará na sua rede local e *vice-versa*. Não há, no nível da aplicação e do usuário final, grande distinção entre sua própria rede local e a Internet, e o NAT é a abstração que torna esta indistinção possível.

3.1 A Abstração

Em um sentido amplo, uma abstração é uma generalização de um conceito, derivado de objetos reais que possuem a propriedade abstraída. Por exemplo, uma determinada cor é uma abstração para descrever um atributo de objetos que refletem um certo comprimento de ondas eletromagnéticas. É um recurso cognitivo que facilita a compreensão e organização das ideias.

Na computação, o termo é muitas vezes utilizado para designar uma simplificação de conceitos e processos: um tipo abstrato de dado "pilha" implica em um conjunto de comportamentos esperados e bem conhecidos de uma "pilha" de dados, mas esconde os processos reais e mais complexos de como obter da memória do computador um comportamento de "pilha". Sendo de utilidade explícita na computação, abstrações são utilizadas sobre abstrações, diminuindo cada vez mais a complexidade e tornando os conceitos mais tratáveis à medida em que o nível de abstração aumenta.

No nível mais alto, as abstrações oferecidas aos usuários finais são em geral muito mais simples que aquelas usadas pelos programadores. Os programas tendem a ser modelados tendo em mente conceitos conhecidos de origem externa aos computadores, de modo que correios deram origem ao correio eletrônico (*e-mail*), máquinas de escrever viraram processadores de texto e pranchetas de desenho deram origem a ferramentas CAD (*Computer Aided Design*). A própria computação, por sua vez, como tudo que influencia

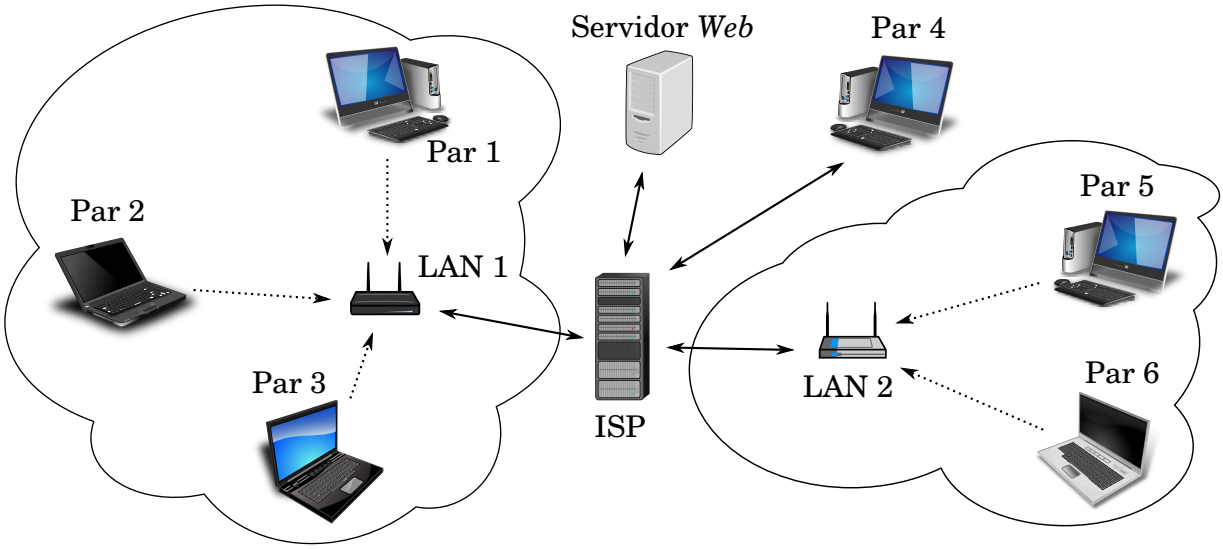


Figura 3.1: Alcançabilidade entre pares através do NAT

a vida das pessoas, originou suas próprias abstrações, como por exemplo, a Internet. Se Internet originalmente era um mecanismo de interligar redes remotas de computadores, o conceito abstrato de Internet se tornou a possibilidade de se comunicar com qualquer *coisa* em qualquer lugar do mundo.

NAT provê uma abstração para esta Internet, e permite sua utilização mesmo que o equipamento esteja tecnicamente em uma rede privada (e não na Internet). Um dos elementos de rede, o *gateway*, está ciente da abstração e esconde a complexidade das traduções de endereços atrás da interface padrão do UDP/TCP e IP. Para muitos usuários e aplicações, o funcionamento ordinário do NAT os coloca efetivamente na Internet, caracterizando bem a abstração. Entretanto, certos tipos de aplicação, as que precisam expor serviços para a Internet, não são funcionais através do NAT. Para que sejam, as próprias aplicações ou os usuários precisam estar cientes do NAT e adotar alguma medida para contornar suas limitações.

Considere a Figura 3.1, apresentando um grafo direcionado onde a alcançabilidade de um nó para outro representa a habilidade deste nó iniciar uma comunicação com o outro através da Internet. Enquanto as necessidades do Par 1 forem somente alcançar o Servidor *Web*, a tradução de endereços que acontece em LAN 1 não será sentida. Também não haverá problemas se o Par 2 quiser fazer *download* de um aquivo do Par 4 através de algum aplicativo P2P, uma vez que, sendo uma iniciativa do Par 2, ele será capaz de estabelecer a conexão TCP necessária com o Par 4. O contrário será um problema. Por sua própria iniciativa, o Par 4 não será capaz de estabelecer a conexão TCP da aplicação P2P com o Par 2, pois o máximo que o Par 4 conseguirá endereçar é o roteador LAN 1. Pior ainda, os Pares 1, 2 e 3 são invisíveis aos Pares 5 e 6 (e *vice-versa*) em um ambiente P2P. O mesmo problema ocorreria caso o Par 3 tentasse se juntar a uma sessão de um jogo sediada pelo Par 6, pois não há nenhum endereço ao qual ele pudesse enviar o primeiro pacote UDP que o colocaria no jogo, já que o endereço IP do Par 6 está mascarado pelo

roteador LAN 2.

Para contornar estes problemas, o roteador deve ser configurado para encaminhar as tentativas de conexão TCP e os pacotes UDP desconhecidos a nós e portas específicos dentro da rede, sendo estes os nós que estão executando as aplicações responsáveis por tratar cada requisição encaminhadas. Deve-se notar que isto não é um simples roteamento IP, já que o endereço de destino do datagrama IP é o do próprio *gateway* (que é um endereço público), e não o do nó da aplicação (que é privado).

A necessidade de cada vez mais aplicações serem alcançáveis externamente começou a enfraquecer a abstração do NAT. O uso generalizado do NAT tornou-o uma preocupação para os usuários e desenvolvedores de tais aplicações. É agora comum encontrar aplicações que implementam protocolos para configurar automaticamente o NAT em equipamentos de *gateway*. Também é comum encontrar usuários avançados de P2P cientes do problema e experientes na configuração manual da passagem de NAT. O propósito de uma abstração deveria ser simplificar, mas se a abstração é frequentemente violada e as complexidades que ela deveria esconder estão expostas, então sua utilidade está comprometida.

No cenário atual temos usuários cientes do NAT, executando aplicações também cientes, sobre sistemas operacionais que a *ignoram*, que se comunicam através de um *gateway* de Internet que implementa a técnica de NAT; técnica esta que tem como objetivo ser invisível. Como notado na documentação oficial do NAT:

> "*Basic Network Address Translation or Basic NAT is a method by which IP addresses are mapped from one group to another, transparent to end users.*"[1] [Srisuresh e Egevang 2001]

Considerando que não é possível oferecer uma solução completa para a abstração da Internet tendo o NAT implementado somente dentro do *gateway*, a abordagem proposta então é que o sistema operacional, sendo responsável pela infraestrutura de rede e estando em um nível de abstração mais baixo, complemente a abstração de tradução de endereços dada pelo *gateway*, aliviando usuários e aplicações de alto nível da necessidade de se preocuparem com ele.

3.1.1 Usabilidade

NAT é a realidade comum para usuários de Internet, e dificilmente seu terminal, seja em casa ou no trabalho, estará diretamente conectado a ela. Na era do P2P, as implicações práticas desta realidade para os usuários finais é cada vez mais sentida. Não é incomum encontrar aplicações que não podem ser usadas, em todo seu potencial, sem que o redirecionamento do NAT seja configurado manualmente no roteador.

[1] "Tradução de Endereço de Rede Básica ou NAT Básica é um método pelo qual endereços IP são mapeados de um grupo para outro, transparente aos usuários finais."

Para usuários avançados, que têm familiaridade com computadores e que através da experiência aprenderam a lidar com o problema, configurar o NAT não passa de um aborrecimento. Esta não é a realidade para a grande maioria dos usuários, que não têm interesse outro além de usar o computador para os seus propósitos, sem se preocuparem com a infraestrutura ou os detalhes técnicos. Estes dependem das abstrações de usabilidade oferecidas pelo sistema.

O conceito de usabilidade vem sendo desenvolvido ao longo dos anos em que o computador deixou de ser uma ferramenta para técnicos especializados para se tornar uma ferramenta de trabalho comum nas empresas, e mais tarde uma ferramenta doméstica. O foco da usabilidade é interação dos usuários finais com a máquina, e procura oferecer um conjunto de abstrações sobre o qual o usuário possa se apoiar, sem não precisar entender os detalhes para usar o computador.

Se o sistema operacional avisa que o computador está conectado à Internet (isto é a realidade dos sistemas operacionais atuais), então não é aceitável que alguns programas deixem de funcionar por uma limitação do NAT. Isto se constitui em uma quebra da usabilidade do sistema: o usuário sabia estar conectado à Internet, mas a aplicação que depende da Internet não funciona, por fatores desconhecidos e alheios à sua vontade. A confiabilidade da abstração conhecida como “conexão com a Internet” é abalada, e o usuário é impossibilitado de utilizar aquela aplicação.

3.1.2 Reusabilidade

Na engenharia de *software*, reusabilidade é um conceito chave para se conseguir redução dos custos de desenvolvimento de um projeto. Consiste essencialmente em reaproveitar partes de *software* que já foram escritas, evitando a necessidade de desenvolver novamente a mesma funcionalidade.

Existem vários níveis de reuso de *software*, em variados níveis de abstração. Uma abordagem comum é a utilização de *frameworks*, os quais são esqueletos de sistemas já pré-fabricados para um domínio de aplicação específico. Estes possuem um nível de abstração muito alto, e oferecem um tempo de desenvolvimento reduzido, mas em compensação estão restritos a um domínio específico e as funcionalidades da aplicação são limitadas às funcionalidades do *framework*.

Em um nível de abstração mais baixo, na orientação a objetos é comum a reutilização de classes e módulos inteiros entre aplicações. É uma abordagem mais flexível que o uso de *frameworks*, porém com custo de desenvolvimento maior: é o preço que se paga pela flexibilidade adquirida. Se as funcionalidades já não estão prontas como estão em um *framework*, é possível tê-las como desejar, mas terá de arcar com o seu de custo de desenvolvimento.

Antes das classes, em um nível de abstração ainda mais baixo, eram as bibliotecas

de sub-rotinas a principal forma de reúso. Elas foram desenvolvidas ao ponto de ficarem disponíveis, dinamicamente em tempo de execução, por meio dos objetos compartilhados do Unix (*shared objects*, arquivos de extensão ".so") e dos arquivos DLL (*Dynamic-link library*) do Windows. Entretanto, originalmente deviam ser compiladas estaticamente nos programas, com exceção de um conjunto muito particular de sub-rotinas: as chamadas de sistema.

Nesta perspectiva, o sistema operacional também é um mecanismo de reúso de *software*, sendo um dos primeiros e mais importantes, pois alivia o programador da necessidade de interagir diretamente com o *hardware* e, mais recentemente, com a rede, provendo uma interface bem definida e comum a todos os programas. Um *driver* de um novo dispositivo acrescentado ao núcleo do sistema operacional permite sua utilização transparente por qualquer programa deste sistema.

É possível notar uma hierarquia nos mecanismos de reúso de *software*, sendo que os *frameworks* utilizam módulos e bibliotecas de rotinas, e os módulos de linguagens de alto nível, por sua vez, utilizam rotinas de bibliotecas da linguagem de programação base do sistema, que é a linguagem C, e que dentre elas figura a interface principal das chamadas de sistema, como funções da biblioteca padrão C (estendida com os padrões POSIX ou outras extensões proprietárias).

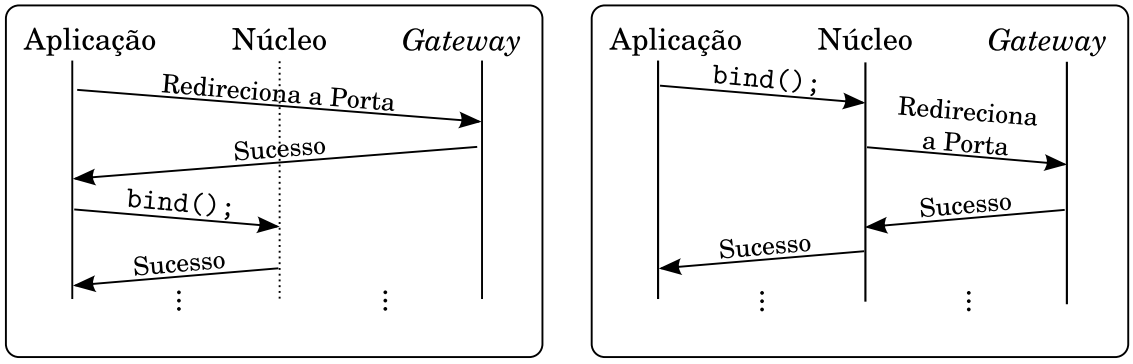
Considerando esta hierarquia, quanto mais básico for o nível de reúso de *software* em que uma nova funcionalidade é adicionada, mais aplicações se beneficiam automaticamente desta funcionalidade. Por exemplo, se um mecanismo de passagem automática de NAT for adicionado a algum *framework* de aplicações em rede como o Twisted[2], somente aplicações escritas com o Twisted se beneficiariam da funcionalidade. Se, por outro lado, o mecanismo fosse implementado no módulo `socket` da linguagem Python, qualquer aplicação de rede feita na linguagem se beneficiaria da funcionalidade, inclusive as aplicações feitas com o Twisted, que é baseado neste mesmo módulo.

3.2 O Método

Para tornar o NAT transparente às aplicações que oferecem seus serviços externamente, é necessário que, usando-se somente da interface *socket* do sistema operacional, uma aplicação consiga aguardar por uma conexão externa, com a mesma simplicidade que uma aplicação cliente se conectaria a um servidor remoto. Assim como usuários e desenvolvedores de aplicações cliente não precisam se preocupar se a máquina opera através de um NAT, desenvolvedores e usuários de aplicações servidoras também não deveriam se preocupar.

Para atingir este objetivo, é preciso tornar os sistemas operacionais cientes do problema

[2]Twisted é um *framework* código aberto na linguagem Python para desenvolvimento de aplicações de rede.



(a) Procedimento original (b) Procedimento proposto

Figura 3.2: Procedimento de passagem de NAT

do NAT, e fazê-los tomar parte na questão. Uma vez que eles já são responsáveis por prover a API de *sockets* – que é usada pelas aplicações para ter acesso ao UDP/TCP/IP, ele possui os meios necessários para gerenciar automaticamente o *gateway* no lugar da aplicação (ou do usuário, nos casos piores).

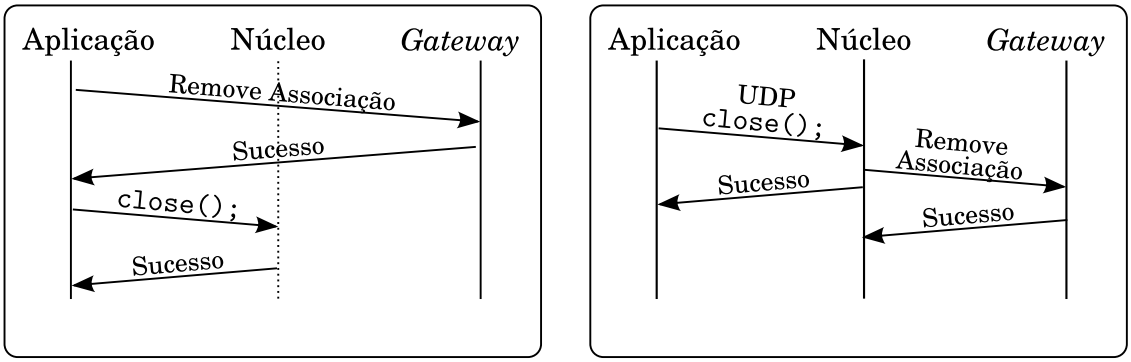
3.2.1 Associação e Encerramento

Ao receber uma chamada de sistemas `bind()` feita em um *socket* TCP ou UDP, o sistema operacional deve usar os mesmos protocolos que as próprias aplicações utilizariam para redirecionar a porta no *gateway*. Caso a porta já esteja ocupada por outro nó na rede, a chamada de sistema deve falhar, de modo que a aplicação consiga executar seu comportamento padrão em caso de uma porta já estar ocupada.

O procedimento para passagem de NAT, que originalmente deveria ser realizado pela aplicação, como mostrado na Figura 3.2a, passa então a ser realizado pelo próprio sistema operacional, como mostrado na Figura 3.2b. Isso modifica um pouco o comportamento esperado da chamada de sistema `bind()`, que originalmente retorna imediatamente. Como as aplicações não estão preparadas para serem bloqueadas nesta chamada, é importante que exista um temporizador que, no caso de algum problema durante a associação do *gateway*, impeça a chamada de permanecer bloqueada por muito tempo.

Quando aquele ponto de comunicação não for mais necessário – e isto é sinalizado quando a aplicação fecha explicitamente o *socket* com a chamada de sistemas `close()`, ou então quando a aplicação é encerrada – o sistema operacional deve remover o redirecionamento correspondente do *gateway*. Vale ressaltar que mesmo que seja um *socket* TCP, este é o *socket* por onde as conexões são estabelecidas, e não o *socket* de alguma conexão TCP específica, portanto nenhuma comunicação propriamente dita é feita através dele, e ele não está sujeito a encerramento por inatividade, como os *sockets* das conexões.

Aplicações que redirecionam automaticamente a porta devem, ao terminar de utilizá-la, remover a sua associação no *gateway*, como mostrado na Figura 3.3a. Entretanto, seja por causa de um erro que encerrou forçosamente a aplicação ou por causa de uma má



(a) Procedimento original (b) Procedimento proposto para o UDP

Figura 3.3: Procedimento de remoção da passagem de NAT

utilização do protocolo de passagem de NAT, a aplicação pode deixar de fazê-lo. Não é possível para o *gateway* determinar que esta situação ocorrera, já que as aplicações operam fora do seu controle. Originalmente, o sistema operacional não pode ajudar em nada neste caso, já que é passivo no processo de redirecionamento de porta, e portanto a associação poderia permanecer ativa no *gateway* por tempo indeterminado. Esta situação é análoga a um vazamento de memória (*memory leak*), onde também há desperdício de recursos.

Um efeito colateral benéfico do método aqui proposto é que, já que a passagem de NAT é automaticamente gerenciada pela infraestrutura de rede no sistema operacional, ele detém o controle da aplicação em execução, e é capaz de remover a associação do *gateway* mesmo em caso de encerramento abrupto da aplicação. Contanto que o sistema operacional continue em funcionamento, ele é cúmplice do *gateway* no processo de tradução de endereços, e não o deixa refém das aplicações e seus problemas.

Outra situação que pode ocorrer quando a passagem de NAT é configurada explicitamente pela aplicação é o não encerramento das conexões TCP estabelecidas. Quando uma aplicação é encerrada ou fecha um *socket* correspondente a uma conexão TCP ativa, o sistema operacional não encerra imediatamente aquela conexão, porém o *socket* é liberado de imediato e a aplicação pode concluir.

Como o TCP é um protocolo orientado a conexão, o sistema operacional deve negociar o encerramento da conexão com o seu par remoto. Isso implica em enviar uma mensagem TCP com *flag* `FIN`, aguardar as mensagens `ACK` e `FIN` do par remoto e finalmente responder com `ACK`.

Caso a porta usada pela comunicação não seja a mesma porta que a aplicação utiliza para receber conexões, não há problema nenhum em remover o redirecionamento antes do encerramento das conexões pendentes. Entretanto, como é o caso da pilha TCP do Linux, pode ser que a porta utilizada para receber as mensagens das conexões seja a mesma utilizada para iniciá-las. Neste caso, o redirecionamento da porta no *gateway* não deve ser removido até que todas as conexões pendentes sejam devidamente encerradas. A não ser que a aplicação aguarde o encerramento de todas as conexões com a opção do

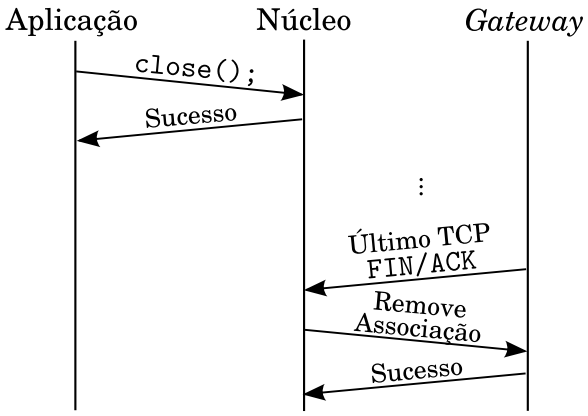


Figura 3.4: Remoção de uma associação TCP com conexões pendentes

socket `SO_LINGER`, ao remover um redirecionamento ela impede que todas as conexões TCP pendentes iniciadas por lá sejam corretamente terminadas.

Este efeito não causa um problema maior de usabilidade, e as conexões serão eventualmente encerradas pelos *timeouts* do TCP. Entretanto, esta é uma situação que poderia ser facilmente evitada caso o núcleo do sistema operacional cuidasse da passagem de NAT. No caso, o encerramento de uma associação de porta TCP não pode ser feito igual ao do UDP, como mostrado na Figura 3.3b, o que, para a conexão pendente, é equivalente ao que uma aplicação faria. Em vez disso, o encerramento da associação deve ser feito como na Figura 3.4, e a associação só é removida do *gateway* quando todas as conexões TCP ativas por aquela porta forem fechadas.

Com a aplicação deste método, levamos o problema da passagem de NAT ao nível mais baixo das categorias de reúso de *software*, que é o sistema operacional, e portanto levamos a funcionalidade ao maior número de aplicações possível, sem a necessidade do desenvolvedor depender de alguma biblioteca ou *framework* de mais alto nível para dispor dela, ou o que seria ainda mais custoso, implementar ele mesmo a funcionalidade.

Já que todas as aplicações sobre API de *sockets* do sistema operacional seriam afetadas, também espera-se resolver o problema da usabilidade explicado na Seção 3.1.1, que consiste em aplicações de Internet, mesmo estando o sistema devidamente conectado, não funcionarem corretamente. Aplicações P2P já existentes que são afetadas pelo problema do NAT passariam a funcionar sem nenhuma modificação, já que a solução foi aplicada um nível abaixo delas, diretamente no sistema operacional.

3.2.2 Considerações de Segurança

O método proposto de passagem automática de NAT tem como alvo as aplicações de usuário final. Por questões de segurança, é importante que *daemons* do sistema e servidores que requeiram controle administrativo fino, como FTP[3], HTTP, Telnet e SSH[4]

[3] *File Transfer Protocol*.
[4] *Secure Shell*.

não sejam expostos automaticamente à rede externa. Por esta razão, tentativas de utilizar portas TCP ou UDP abaixo da 1024 devem ser filtradas e não configuradas no roteador. Os serviços previamente mencionados tentam, por padrão, usar portas desta faixa, e é necessário ter privilégios de administrador para tal. Como nenhuma aplicação de usuário comum deve tentar utilizar uma porta privilegiada, sua utilização está fora do escopo deste trabalho.

Aplicações devem especificar qual das interfaces IP disponíveis no sistema usar ao executar o `bind()`. O endereço especial 0.0.0.0 (chamado de `INADDR_ANY` nos sistemas POSIX) quando usado neste contexto, significa que todas as interfaces disponíveis serão utilizadas. Aplicações que não utilizam este endereço genérico e, em vez disso, especificam alguma interface válida, normalmente têm conhecimento de seus potenciais pares e detêm um controle fino da topologia da rede. Neste caso a associação ao endereço/porta não é considerada genérica o suficiente para ser automaticamente visível através do *gateway* de Internet, mesmo que o endereço especificado seja a rota para ele.

Aplicações de Internet não tentam restringir sua alcançabilidade. Se elas devem ser vistas na Internet, sua escolha lógica de por qual interface IP ela é acessível deve ser *qualquer uma* – ou seja, 0.0.0.0. Se o programador ou o usuário quiser controlar sua conectividade ao escolher qual interface utilizar, então este controle não deve ser tirado com a exposição automática da aplicação para a rede externa.

É possível questionar que, sendo a tarefa de abrir portas no roteador automática, o sistema estaria mais vulnerável a vírus e programas maliciosos, pois sendo este método implementado no sistema operacional, um vírus seria capaz de expor o sistema para a rede externa, que de outra forma estaria exposto somente à rede interna. Este não é bem o caso, e um vírus poderia encontrar seu caminho através do NAT da mesma forma que um programa legítimo faria, bastando implementar o mesmo protocolo de passagem de NAT usado por eles, que funciona sem autorização especial. Neste aspecto, portanto, um sistema com o método não seria menos seguro que um sistema sem ele.

3.2.3 Limitação

O mecanismo proposto não resolve totalmente o problema da transparência do NAT. Além do fato de não expor para a Internet as portas abertas por uma aplicação da rede interna, o NAT impõe outra limitação que pode ferir a abstração da Internet: a não ser que cuide explicitamente do NAT, uma aplicação não tem como saber qual o seu endereço IP visível externamente, e se o protocolo da aplicação utilizar seu endereço IP dentro do *payload*, durante a comunicação, ele será o endereço IP da rede privada, que não tem utilidade na Internet. Isto impede que estas aplicações funcionem corretamente.

Esta fora do escopo deste trabalho resolver este problema, porém como trabalho futuro será apresentada na Seção 5.1.1 uma possível extensão deste trabalho que abrangeria

também uma solução para ele.

Capítulo 4

Implementação de Referência

Como prova de conceito da abordagem descrita neste trabalho, foi desenvolvida uma implementação de referência. Ela foi desenvolvida sobre o sistema operacional GNU/Linux, mais especificamente, sobre distribuição Ubuntu. Como mecanismo de passagem de NAT, foi utilizado o UPnP IGD.

O motivo da escolha do protocolo UPnP IGD, como já mencionado na Seção 2.4.2, é devido às suas popularidade e disponibilidade. É a solução para passagem de NAT disponível nos produtos prontamente encontrados no mercado e usados nas pequenas redes locais.

Por necessitar de intervenções abaixo da interface de *sockets*, no interior da pilha de rede, foi necessário para esta implementação ter acesso ao núcleo do sistema operacional. O GNU/Linux foi escolhido por ser o mais popular dos sistemas operacionais livres, onde o código-fonte do núcleo (o Linux propriamente dito) é público e sua licença de uso permite que ele seja estudado e alterado, condições necessárias para este trabalho.

As escolhas das tecnologias UPnP IGD e GNU/Linux influenciaram a arquitetura e as decisões tomadas durante o desenvolvimento desta implementação de referência. Entretanto, nada impede que a abordagem para transparência de NAT descrita na Seção 3.2 seja utilizada com outras tecnologias em outros sistemas operacionais. Em uma possível implementação que utilize tecnologias diferentes, a arquitetura aqui descrita pode não ser a mais adequada.

4.1 Arquitetura

Inspirado pela arquitetura de um *microkernel*, na qual funções inerentes ao sistema operacional são realizadas por processos especiais isolados, esta implementação se divide basicamente em duas partes. Uma é parte do núcleo, que é responsável por interceptar as ações do usuário na interface *sockets* e repassar os eventos à outra parte. Esta outra consiste em um processo de usuário que faz a interface com o *gateway*, implementando o protocolo UPnP IGD. A Figura 4.1 destaca, em cinza, estas partes da arquitetura.

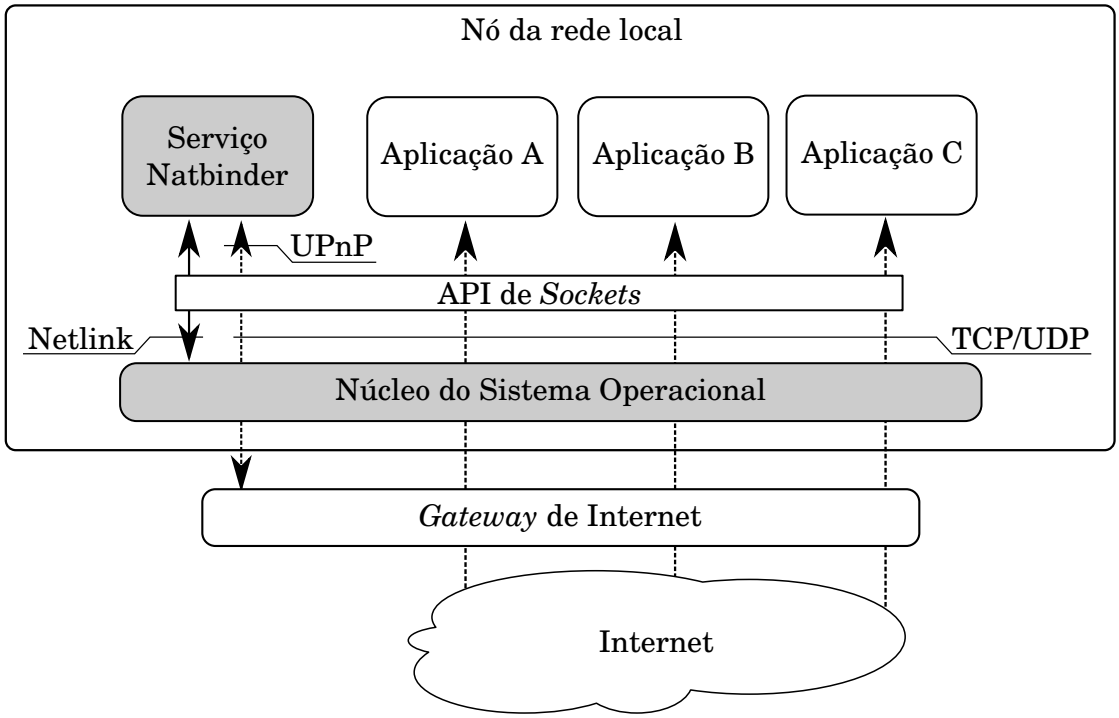


Figura 4.1: Arquitetura da implementação de referência

O motivo de se ter uma arquitetura dividida, e tratar a comunicação com o *gateway* em um processo isolado, é a complexidade do protocolo responsável por esta comunicação, que pertence à camada de aplicação. As tecnologias em que os protocolos UPnP são baseados, tais como XML e SOAP[1], são de alto nível e suas implementações já existentes nas bibliotecas de *software* só podem ser usadas em espaço de usuário, o que inviabiliza a implementação do UPnP IGD dentro do *kernel*.

A própria necessidade de se utilizar o TCP na configuração do *gateway* contribui para a escolha, pois se por um lado existe uma interface muito bem definida para o TCP em nível de usuário, por outro lado dentro do *kernel* sua interface é volátil, mal documentada, e não pode ser garantida no rápido ritmo de evolução do Linux.

4.2 Núcleo do Sistema Operacional

Dentro do núcleo do sistema operacional, toda invocação da chamada de sistema POSIX `bind()` feita em um *socket* IPv4 que usa TCP ou UDP é interceptada. Como discutido na Seção 3.2.2, se a chamada envolver portas privilegiadas ou apontar uma interface de rede específica, ela é ignorada. Do contrário, é um caso para redirecionamento no NAT: os pacotes destinados àquela porta que chegarem no *gateway* de Internet devem ser redirecionados para a aplicação que invocou o *bind*, em um computador interno à rede.

O núcleo, então, repassa a requisição do *bind* para o *daemon* em espaço de usuário responsável por configurar a passagem de NAT. O processo original, usuário da rede, é

[1]XML está mais próxima de uma sintaxe abstrata do que de uma sintaxe de transferência.

tirado de execução colocado em estado de espera (*sleep*) enquanto o núcleo aguarda a resposta do *daemon*. Quando este responder, o processo original é acordado para lidar com a resposta. A chamada de sistema `bind()` pode então ser completada, falhando ou sucedendo de acordo com a resposta recebida.

Processos em espera, aguardando pela resposta do *daemon*, são colocados em uma lista encadeada. Quando o *daemon* envia uma resposta, esta lista é percorrida até o processo pertinente ser encontrado. Não há garantia explícita de que a primeira requisição será a primeira respondida pelo *daemon*, mas este é o cenário provável, e o mecanismo de comunicação utilizado não dá indícios de que poderia ser diferente. Já que os processos são enfileirados na lista na mesma ordem em que suas requisições são enviadas ao *daemon*, quando uma resposta chegar ela provavelmente será referente ao primeiro processo da lista (o que espera a mais tempo), fazendo da busca do processo na lista, na prática, uma operação de tempo constante.

4.2.1 Condição de Corrida Inerente

Há uma condição de corrida inerente ao se alocar uma porta para um processo tanto na pilha de rede do sistema operacional quanto na tabela de redirecionamento de portas externa, contida no *gateway*. O procedimento para se alocar, individualmente, uma porta em cada um destes espaços de nomes pode ser atômico, mas o procedimento para alocá-la em *ambos* não o é.

Considere o seguinte cenário: o processo A tenta, via `bind()`, utilizar o endereço 0.0.0.0 na porta 6881, e é colocado em espera enquanto o *daemon* não responde. Logo a seguir, o processo B tenta utilizar o endereço 127.0.0.1 na porta 6881. Esta requisição pode ser atendida imediatamente, antes da requisição do processo A, pois ela especifica uma interface de rede, a de *loopback*, o que não ativa o mecanismo de passagem de NAT e, portanto, não requer uma negociação custosa via rede. Neste cenário, a qual processo será dada a porta 6881?

A primeira alternativa para lidar com tal situação seria uma abordagem pessimista, e adiar a alocação da porta internamente para o processo A até que o *daemon* tenha reportado sucesso ao tentar configurar aquela porta no *gateway*. Somente depois a porta seria alocada para A na pilha de rede. No nosso cenário, enquanto A esperasse pela resposta, a porta seria alocada para B antes que o *daemon* pudesse responder. Se o redirecionamento da porta falhasse, a requisição feita pelo processo A falharia sem nenhum outro problema. Por outro lado, se o *daemon* tivesse sucesso ao redirecionar a porta, ela não estaria mais disponível localmente para o processo A, então o `bind()` teria de falhar e o *kernel* teria de requisitar ao *daemon* a remoção do redirecionamento recém criado.

A segunda alternativa seria usar uma abordagem otimista, e pré-alocar a porta para A

antes de encaminhar a requisição para o *daemon*. No cenário de exemplo, a porta estaria indisponível para B, e sua requisição de `bind()` falharia. Se a resposta do *daemon* for sucesso, a requisição de A simplesmente é bem sucedida. No caso de falha em configurar o roteador, os recursos ocupados na pré-alocação da porta são liberados, tornando-se disponíveis para outro processo. Esta é, então, a desvantagem desta alternativa: existe uma janela de tempo, de no máximo meio segundo (tempo arbitrariamente escolhido para este protótipo), enquanto o *kernel* espera pela resposta do *daemon*, quando a porta estará indisponível para processos que tentem utilizá-la em interfaces específicas, enquanto há a *possibilidade* de que ela fique disponível novamente após essa janela de tempo.

Finalmente, existe a abordagem correta, pois consiste em tornar atômico todo o processo de alocação de porta para o processo A. Enquanto o futuro de uma porta específica está nas mãos do *gateway* externo, no máximo um outro processo que queira usar aquela porta em uma interface específica deve também aguardar pela resposta do *daemon*. No nosso cenário, caso o *daemon* reporte sucesso, a porta é entregue ao processo A, caso contrário, é entregue ao processo B, que também estaria em estado de espera. Não há necessidade de fazer aguardar mais de um processo extra, porque se a porta não for dada ao A, ela será necessariamente dada ao B. Qualquer outra tentativa de `bind()` naquela porta poderia falhar imediatamente.

Supõe-se que uma resposta positiva do *daemon* seja o caso mais comum, o que torna a primeira alternativa inviável. Também supõe-se que um cenários tal qual o dado no exemplo, onde um serviço de Internet está competindo com um serviço local pela mesma porta ao mesmo tempo, é muito raro. Portanto, acredita-se que utilizar a segunda alternativa, mesmo com suas desvantagens, compensa mais do que a complexidade adicional de implementação necessária à terceira alternativa, que é a mais correta. Assim, a segunda alternativa é a utilizada nesta implementação de referência.

4.2.2 Comunicação Entre *Kernel* e Modo Usuário

Como a arquitetura é dividida em duas partes, uma no núcleo do sistema operacional e outra em modo usuário, é necessário um mecanismo que permita a comunicação entre as duas partes. Um requisito básico deste mecanismo de comunicação é que o *kernel* deve ser

Tabela 4.1: Requisitos dos Métodos de Comunicação *Kernel* ⇔ Modo Usuário

Método	Ativado pelo *Kernel*	Bidirecional	Dados Arbitrários
Chamadas de Sistema	*Não*	*Não*	**Sim**
Sinais	**Sim**	*Não*	*Não*
Via Sistema de Arquivos	**Sim**	**Sim**	**Sim**
Via *Sockets*	**Sim**	**Sim**	**Sim**

capaz de iniciar a comunicação, de modo que possa encaminhar as requisições de `bind()` relevantes de imediato, sem precisar esperar por uma iniciativa do processo do *daemon*. Além disso, o *daemon* precisará responder às requisições e, portanto, a comunicação deverá ser bidirecional. Como os parâmetros do `bind()` devem ser passados na requisição, o método de comunicação deve permitir o envio de dados arbitrários, e não só de mensagens já predefinidas.

O Linux provê uma grande variedade de métodos de comunicação entre espaço de *kernel* e espaço de usuário. Os mais básicos dentre eles são os sinais e as chamadas de sistema. Efetivamente, qualquer outro método de comunicação núcleo ⇔ usuário é feito através de alguma chamada de sistema, sendo as chamadas de sistema também, elas próprias, mecanismos de comunicação.

A Tabela 4.1 relaciona os requisitos com os métodos de comunicação disponíveis, onde é possível ver que sinais e chamadas de sistema, por não atenderem aos requisitos, não podem, por si só, ser usados para este propósito. Mesmo se fosse possível usá-los para implementar a arquitetura, seria muito invasivo adicionar novos sinais e chamadas de sistema ao *kernel* somente para este propósito, que é muito restrito comparado às tarefas de propósito geral realizadas pelas chamadas de sistema e sinais existentes.

Em relação aos métodos de comunicação de mais alto nível, é possível dividi-los em duas categorias: os métodos baseados no sistema de arquivos e os métodos baseados em *sockets*. Métodos via sistema de arquivos usam o VFS[2] como ponto de partida e espaço de nomes para a comunicação. Para utilizá-los, um processo usaria as mesmas chamadas de sistema que são usadas para ler e escrever arquivos, em particular `open()`, `read()`, `write()`, `close()`, dentre outras.

Por sua vez, os métodos de comunicação via *sockets* usam da infraestrutura de rede do núcleo, então cada um possui o seu próprio espaço de nomes e podem ser usados através das chamadas de sistemas referentes a *sockets*, como `socket()`, `recvmsg()` e `sendmgs()`.

Como visto na Tabela 4.1, ambas as categorias atendem aos requisitos e, então, o método utilizado foi escolhido pelo seu escopo de utilização mais comum, tendo em vista uma integração natural desta arquitetura com o resto do sistema.

Dentre os métodos de comunicação via VFS, o mais antigo e proeminente são os arquivos especiais de dispositivos localizados no diretório `/dev`. Eles não são uma boa escolha pois, mesmo que alguns deles sejam puramente virtuais, como `/dev/random`, `/dev/urandom` e `/dev/loop0`, eles foram feitos como interface para dispositivos de *hardware* reais conectados ao sistema. Uma interceptação da chamada de sistemas `bind()`, como é o caso aqui, não se configura como um dispositivo propriamente dito para merecer figurar entre os outros dispositivos.

O ProcFS [Bowden et al. 2009] é outro método baseado no sistema de arquivos. Seus arquivos especiais de comunicação estão localizados no diretório `/proc` e são usados

[2]Do inglês, *Virtual File System*, sistema de arquivos virtual.

principalmente para recuperar informações sobre o estado dos processos em execução no sistema. Uma vez que a abordagem atende eventos específicos (os eventos de *socket* IPv4 `bind()` e `close()`) em vez de monitorar as mudanças de estado dos processos, o ProcFS não parece uma boa escolha. Além disso, poucos arquivos virtuais em */proc* permitem escrita por parte do usuário. Os poucos que permitem são usados para definir o estado de algumas configurações, e não para enviar mensagens.

Similar em espírito ao ProcFS, o SysFS [Mochel e Murphy 2010] também é usado para prover informações de estado, mas ao invés de dar as informações referentes aos processos, ele provê informações do estado do próprio sistema, incluindo informações sobre dispositivos de *hardware*. A possibilidade de escrita de alguns de seus arquivos virtuais é usada para configuração, não para envio de dados. Se não cabe utilizar o ProcFS aqui, o SysFS é igualmente inadequado.

Dos métodos de comunicação via *socket*, existe uma API dentro do núcleo que pode ser utilizada para comunicação via UDP com os processos, tanto locais quanto remotos. Se nesta implementação estivesse sendo utilizado o protocolo NAT-PMP para configuração do *gateway*, esta provavelmente seria a melhor escolha. Sem a necessidade de nenhum *daemon* auxiliar em nível de usuário, de dentro do *kernel* seria possível conversar com o *gateway* e configurá-lo. Mas como a necessidade aqui é conversar com um processo local, e este método implica na utilização de um protocolo de rede real, ele não é a escolha mais adequada.

Netlink [Linux Foundation 2009] é outro mecanismo baseado em *sockets*. É um protocolo específico do Linux que, apesar de ser construído sobre sua infraestrutura de rede, não é um protocolo de rede. Processos podem usá-lo para se comunicar com o *kernel* ou outros processos na mesma máquina. Ele é utilizado, entre outras coisas, por muitos serviços de rede providos pelo Linux, tanto para configuração quanto para transferência de dados de rede entre o núcleo e o nível de usuário.

Dada a similaridade dentre a utilização atual do Netlink e as necessidades de comunicação desta implementação, sendo capaz de atender a todos os requisitos, e ainda incluindo uma funcionalidade de *broadcast* de mensagens muito útil para a implementação, ele foi o mecanismo de comunicação escolhido. Através da API do *kernel* para o Netlink, foi criado sobre ele um protocolo específico para os propósitos deste trabalho, chamado `NETLINK_NAT_PASS`.

4.3 *Daemon* Auxiliar

O *daemon* chamado *Natbinder* foi desenvolvido neste trabalho com o propósito de controlar o *gateway* de Internet com o protocolo UPnP IGD, atendendo às requisições do núcleo do sistema operacional. Construído na linguagem C, foi utilizada a implementação do protocolo UPnP IGD dada pela biblioteca de rotinas MiniUPnPc [Bernard 2011].

Como um *daemon* de sistema, é esperado que o Natbinder seja iniciado no procedimento de *boot* do sistema operacional, junto com os outros serviços do sistema. Ao ser iniciado, o Natbinder procura na rede local pelos dispositivos UPnP, identificando o *gateway* dentre eles. Além disso, ele descobre qual o endereço IP local da máquina na rede do *gateway* e usa esse endereço para construir as requisições UPnP.

Após verificar que é capaz de alcançar o *gateway*, o *daemon* se registra no canal de *broadcast* 0 do protocolo `NETLINK_NAT_PASS` através de um *socket* Netlink. Por este canal ele aguardará as mensagens do *kernel* referentes às associações de portas TCP e UDP em IPv4 feitas pelos processos.

A cada tentativa de `bind()` recebida via Netlink, a ação UPnP *AddPortMapping* é requisitada ao *gateway*. Dentre as respostas para esta requisição especificadas pelo padrão UPnP, duas são de particular interesse: o código 0, que significa sucesso, e o código 718 (*ConflictInMappingEntry*), que significa, como explicado na Seção 2.4.1, que a porta requisitada no `bind()` já está em uso por outro nó da rede local. Nestes casos, as respostas devolvidas ao *kernel* são, respectivamente, para proceder ou falhar com o `bind()`.

Caso o *gateway* devolva uma resposta diferente, ela é tratada como condição excepcional, à qual o sistema não está preparado para lidar e não pode mais ajudar no processo de associação da porta. Neste caso, a mensagem enviada de volta ao *kernel* é de que o Natbinder ignorou aquela requisição. O efeito prático é o mesmo que processado com sucesso, uma vez que impedir o processo de utilizar a porta neste caso não trará nenhum benefício. Um registro com a condição excepcional é gravado para possibilitar a posterior investigação do ocorrido pelo administrador do sistema.

4.4 O Protocolo

O processo de criação de um novo protocolo que funciona sobre o Netlink é simples, sendo necessário apenas escolher um número de protocolo livre no arquivo de cabeçalho `netlink.h` e definir o nome `NETLINK_NAT_PASS` como uma *macro* para este valor. Portanto, a maior parte do trabalho em criar o protocolo reside na definição de seu vocabulário e das interações comportamentais entre o *kernel* e o *daemon*.

Existem dois tipos de mensagens no `NETLINK_NAT_PASS`, as *requisições* e as *respostas*. As requisições são sempre enviadas pelo *kernel*, e podem ser:

- `PORT_BIND`;
- `PORT_CLOSE`.

Seus parâmetros são os seguintes:

- um número de sequencia, que identificará aquela requisição dentro do *kernel*;
- o endereço IP da requisição (deve ser 0.0.0.0 para ser relevante);

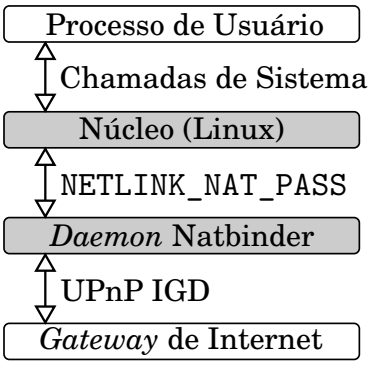


Figura 4.2: Visão lógica das camadas de comunicação

- o número da porta requisitada (deve ser maior ou igual a 1024 para ser relevante);
- o protocolo de transporte utilizado (TCP ou UDP).

Em resposta a uma requisição feita pelo *kernel*, o *daemon* sempre enviará uma das seguintes mensagens:

- `SUCCEEDED`;
- `FAILED`;
- `IGNORED`.

Sendo do tipo *resposta*, cada mensagem leva como parâmetro o número de sequência da requisição correspondente, de modo que o *kernel* possa relacionar cada resposta recebida com uma requisição pendente.

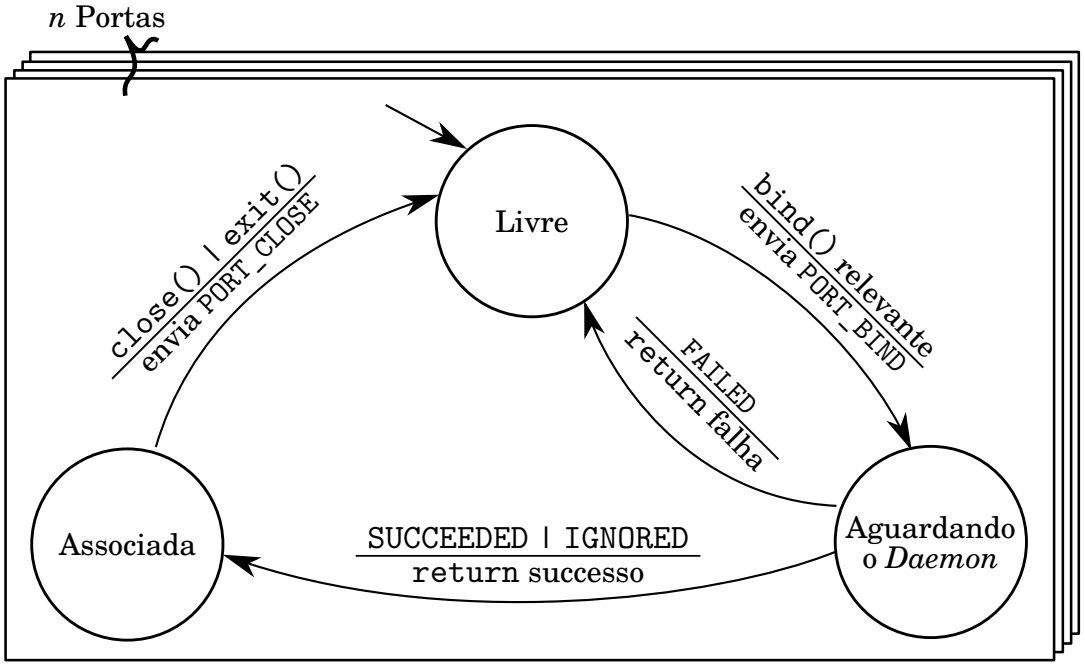


Figura 4.3: Máquina de estados das portas do *Kernel*

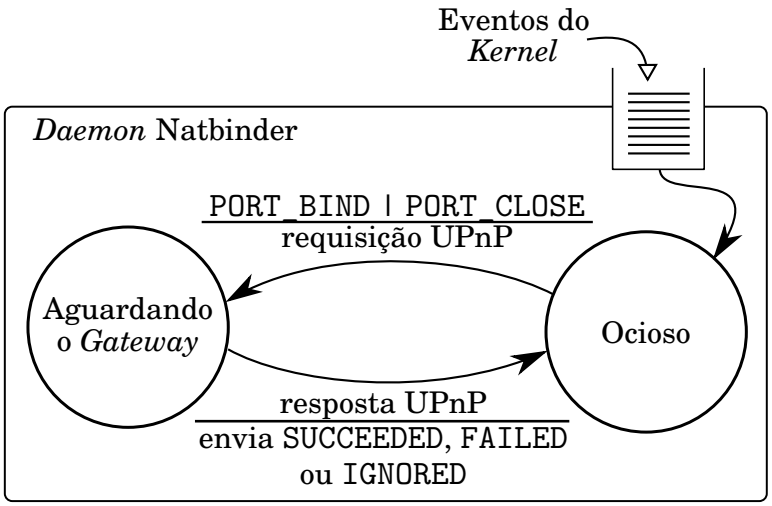


Figura 4.4: Máquina de estados do *daemon* Natbinder

Toda a dinâmica desta comunicação pode ser vista como uma arquitetura em camadas, como mostrado na Figura 4.2. Nesta perspectiva, a camada superior são os processos de usuário, que são servidos pelo *kernel*. Este provê seus serviços aos processos através das chamadas de sistema, e por sua vez é servido pelo *daemon* Natbinder, na camada inferior. Nesta camada, o protocolo criado `NETLINK_NAT_PASS` é a interface que o *daemon* usa para prover seus serviços ao *kernel*. Na última camada, o *gateway* de Internet serve ao Natbinder através do protocolo UPnP IGD.

Foi implementado dentro do *kernel* um autômato de três estados para cada porta (veja Figura 4.3). Este autômato reage a eventos de chamadas de sistemas relevantes ao redirecionamento de portas (como discutido na Seção 3.2.2), e requisita os serviços do *daemon*. Um evento `bind()` ativa a requisição de um `PORT_BIND` ao *daemon*, para que este realize o redirecionamento automático da porta. Enquanto isso, o autômato se mantém em espera no estado "Esperando pelo *Daemon*" até a chegada da mensagem de resposta (ou o prazo máximo se esgotar). Quando a porta não for mais utilizada, e a mensagem `PORT_CLOSE` for enviada, o autômato não espera resposta, voltando imediatamente ao estado "Livre".

No *daemon*, as requisições do *kernel* são seriadas pelo protocolo Netlink, sendo tratadas sequencialmente. Como mostrado na Figura 4.4, para cara mensagem tipo *requisição* que chega, uma mensagem tipo *resposta* correspondente é enviada de volta ao *kernel*, utilizando o mesmo número de sequência da requisição.

4.5 Recuperação de Falhas

É muito importante que uma aplicação não fique muito tempo à espera da conclusão da chamada *bind()*. Uma vez que o *kernel* não detém o controle do *status* dos processos escutando no canal de *broadcast* do `NETLINK_NAT_PASS`, duas precauções são tomadas

dentro do *kernel*. Primeiro, antes de enviar a requisição de `PORT_BIND`, o *kernel* verifica se há algum processo escutando no canal de *broadcast*. Se não houver, o procedimento ordinário para associação de endereço é realizado.

Se por alguma razão o Natbinder não estiver em execução, seja por culpa de um *bug* ou porque o processo foi explicitamente desligado pelo usuário, esta verificação por parte do *kernel* vai garantir que a chamada `bind()` seja executada normalmente (isto é, sem a funcionalidade de passagem automática de NAT).

Também pode haver o caso em que a requisição de `PORT_BIND` já tenha sido enviada pelo *kernel*, mas por alguma razão, seja um *bug* ou erro de rede, o *daemon* parou de responder. Para evitar que o processo de usuário fique bloqueado em espera indefinidamente, dentro do *kernel* é acionado um temporizador de meio segundo. Após esse tempo, a chamada `bind()` é concluída como se a resposta recebida tivesse sido a mensagem `IGNORED`.

Ainda que o *daemon* esteja corretamente em execução, o *gateway* não é totalmente confiável. As configurações feitas pelo Natbinder são voláteis, então basta que o usuário reinicie manualmente um modem que esteja funcionado como *gateway* para que todas as associações de NAT feitas previamente sejam perdidas. Além disso, as implementações do UPnP IGD dentro dos dispositivos podem ter suas próprias limitações. Por exemplo, um dos dispositivos utilizados na realização deste trabalho tinha um limite máximo de 32 portas encaminhadas via UPnP IGD. Requisições para mapeamento de porta após este limite falham com um código de erro desconhecido, que gera a resposta `IGNORED` para o *kernel*.

Para tratar deste tipo de problema, que diz respeito à confiabilidade do *gateway*, o *daemon* mantém o estado ideal de todas as portas que ele gerencia. Periodicamente, ele verifica o estado destas portas no *gateway* e compara com a sua versão ideal dos estados. Se forem divergentes, o *daemon* tenta realizar as mudanças necessárias no *gateway* para que seu estado se equivalha ao ideal. Assim, se uma associação de porta não for realizada por culpa de algum erro desconhecido, mas idealmente ela deveria ter sido feita, então ela será criada no estado interno de referência. Quando da verificação periódica do *gateway*, esta associação pendente será tentada novamente. Considerando o caso previamente mencionado onde o *gateway* é reiniciado, o *daemon* não encontraria nenhuma porta mapeada nele, então registrá-las-ia todas novamente.

4.6 Testes de Validação

Para testar a abordagem, foram selecionadas três aplicações já preexistentes que são afetadas pelo problema da transparência do NAT, mas de maneiras diferentes. São elas os clientes P2P rTorrent e Transmission e o jogo em rede OpenArena.

O rTorrent[3] é um cliente com interface visual em modo texto do protocolo de distribuição

[3]Disponível em `http://libtorrent.rakshasa.no/`.

de arquivos BitTorrent. Como requisito deste protocolo, ele deve disponibilizar uma porta TCP na Internet por onde receber as conexões dos outros pares. Ele não possui um mecanismo de passagem de NAT e, portanto, originalmente ele não é capaz de escutar por conexões além da rede local privada, a não ser que o roteador seja manualmente configurado.

Já o Transmission[4], que também é um cliente do mesmo protocolo BitTorrent, possui já embutido um mecanismo de passagem de NAT, utilizando para tanto o protocolo UPnP IGD. O Transmission possui suporte a uma extensão do BitTorrent que utiliza para transporte de dados o protocolo μTP [Norberg 2010], criado em cima do UDP, o que faz com que ele, além de uma porta TCP, também necessite receber dados da Internet em uma porta UDP.

O OpenArena[5] é um jogo de ação/tiro em primeira pessoa, que permite que até 16 jogadores participem simultaneamente de uma mesma sessão de jogo. Por ser um jogo em tempo real e requerer controle fino das latências da comunicação, o protocolo do OpenArena foi desenvolvido sobre o UDP. Qualquer usuário pode iniciar e servir a sua própria sessão de jogo, procedimento que abre uma porta UDP na pilha de rede do sistema por onde os outros jogadores se juntarão à sessão, e as mensagens serão transmitidas.

Os testes foram executados em um computador doméstico de arquitetura Intel, ligado a uma rede privada *wireless* por um ponto de acesso D-Link DIR-635, que também faz o papel de *gateway* com tradução de endereços ligado à Internet. O computador executava um sistema operacional Ubuntu modificado com a implementação de referência.

4.6.1 Casos de Teste

Em todos os casos de testes foi esperado que as portas utilizadas pelas aplicações executadas tornassem-se automaticamente visíveis externamente. Além disso, o primeiro e o segundo casos de teste tiveram como objetivo específico verificar o comportamento de processos que concorrem pela mesma porta, tanto localmente no nó quanto no *gateway* de Internet. Pelo fato do *bind()* ter sido modificado para falhar também no caso de a porta requisitada estar indisponível no *gateway*, espera-se que a aplicação consiga lidar com uma porta já utilizada por outra máquina da rede do mesmo modo que lidaria com uma porta ocupada localmente.

No primeiro teste, com o rTorrent, o programa foi configurado para utilizar as portas TCP dentro da faixa de 10000 até 10009. Desta faixa, as de número ímpar foram manualmente ocupadas no *gateway*, simulando sua utilização por outro nó, de modo que só restariam disponíveis as cinco portas pares. Foram iniciadas cinco instâncias do programa, e todas as cinco conseguiram executar corretamente, sendo que cada uma se associou

[4]Disponível em `http://www.transmissionbt.com/`.
[5]Disponível em `http://openarena.ws/`.

a cada uma das portas restantes. Todas as cinco instâncias foram capazes de receber conexões da Internet. A partir da sexta instância simultânea iniciada, as execuções falham com a mensagem de erro: *"Could not open/bind port for listening: Address already in use."*[6]

Aparentemente o rTorrent seleciona sua porta de operação aleatoriamente dentro da faixa permitida, e caso esta falhe, tenta as portas restantes até encontrar alguma usável, ou, caso nenhuma esteja disponível, desiste com a mensagem de erro. Como esperado, através da utilização do método aqui proposto, este mecanismo do rTorrent para lidar com portas já ocupadas localmente se mostrou igualmente útil também com as portas ocupadas no *gateway*.

Como segundo teste, foi executado o servidor dedicado do OpenArena, que por padrão espera pela conexão dos jogadores na porta UDP 27960. Então o jogo foi executado, e dentro dele foi criada uma nova sessão de jogo multijogador, que utilizou a porta UDP 27961. Ambas as portas foram corretamente expostas à Internet, e clientes remotos conseguiram se juntar a ambas as sessões de jogo. Cada associação foi corretamente removida do *gateway* quando o jogo e o servidor dedicado foram encerrados.

Semelhante ao rTorrent, o OpenArena também possui um mecanismo para selecionar qual porta UDP utilizar, caso a porta padrão 27960 não esteja disponível: ele simplesmente utiliza a próxima porta disponível. Cada sessão de jogo *online* necessita de uma porta. Como esperado, a execução do servidor dedicado criou uma sessão de jogo na porta padrão, e quando uma segunda sessão foi criada pelo cliente, esta utilizou a porta seguinte.

O terceiro teste foi realizado com o Transmission, uma aplicação que já utiliza UPnP IGD para implementar passagem de NAT. Teve como um objectivo específico testemunhar o problema mencionado na Seção 3.2.1, onde uma aplicação pode deixar de remover sua associação no *gateway* quando terminada, para daí então comparar com a execução do mesmo programa utilizando o método aqui proposto.

Outro objetivo específico do terceiro teste foi verificar o comportamento do mecanismo de passagem automática de NAT quanto utilizado em conjunto com aplicações que já implementam esta funcionalidade. Como tanto a aplicação quanto o Natbinder tentariam redirecionar a mesma porta para o mesmo nó da rede local, não haveria nenhum problema e a redireção simplesmente funcionaria. As requisições UPnP IGD redundantes feitas ao *gateway* deveriam, de acordo com o protocolo, ser ignoradas.

O terceiro teste foi executado em duas partes. Primeiro, em um sistema Ubuntu normal sem as modificações para passagem automática de NAT. O Transmission foi iniciado, e então morto com um sinal `SIGKILL`, que o terminou abruptamente, simulando uma falha. Na segunda parte do teste, foi utilizado o sistema modificado, onde o Transmission foi executado, e então morto do mesmo modo que na primeira execução.

Como esperado, não ouve conflito com as implementações simultâneas do UPnP IGD

[6] "Não foi possível abrir/associar porta para escuta: Endereço já em uso."

da aplicação e do Natbinder: em ambas as execuções a porta que a aplicação utilizou foi redirecionada pelo *gateway*, e em ambos os casos a aplicação foi capaz de receber conexões externas. Também como esperado, após o encerramento abrupto da primeira execução – no sistema original – o redirecionamento permaneceu ativo no *gateway*, enquanto que na segunda execução – no sistema modificado – o redirecionamento foi automaticamente removido quando aplicação foi morta.

4.7 Distribuição

A implementação de referência, nos moldes do Linux e do sistema operacional GNU, é um *software* livre distribuído sob a licença GNU *General Public License* (GPL), versão 2 [FSF 2012]. Ele está disponível no endereço `http://gitorious.org/natbinder`.

Capítulo 5

Conclusão

Neste trabalho o problema da passagem de NAT foi abordado como um problema de usabilidade que feria uma abstração maior que era a Internet. Dada esta perspectiva do problema, foi proposta uma nova abordagem de trato com o NAT, com a premissa de que ele deveria ser transparente para as aplicações e automaticamente gerenciado pelas camadas inferiores do sistema.

Foi criado um método para atingir esta transparência, que visava integrar às aplicações e interfaces do sistema operacional já existentes um sistema automático de passagem de NAT. Daí foi determinado como este deveria interagir com as chamadas de sistema de *sockets* POSIX, de modo a se encaixar naturalmente no procedimento comum em que as aplicações utilizam-nas.

Tendo as interações comportamentais sido definidas, o método foi implementado em um sistema operacional baseado em Linux, utilizando o protocolo UPnP IGD para passagem de NAT. A abordagem foi então testada e validada, através desta implementação de referência, utilizando aplicações já existentes que sofriam com o problema da passagem de NAT.

Resultado do trabalho desenvolvido, um artigo publicado no *International Conference on Networks* 2012 [Vella et al. 2012]. Com este trabalho, espera-se ter mostrado a viabilidade da abordagem, possibilitando que ela seja implementada nos sistemas operacionais de produção distribuídos em larga escala, como Ubuntu, Windows, iOS, Android, Playstation e muitos outros, onde o impacto social da melhora da usabilidade seria sentido.

5.1 Trabalhos Futuros

Relativo à implementação de referência, esta ainda precisa ser polida para alcançar o *status* de produto final. Como trabalho futuro, já que a implementação foi feita no Ubuntu, ela deve ser inicialmente distribuída no sistema de distribuição de pacotes de terceiros do Ubuntu, o PPA (*Personal Package Archives*), onde poderá alcançar um grande público de

usuários. Quando estiver madura o suficiente, dependendo da aceitação da comunidade, poderá então ser inclusa na distribuição oficial.

Como uma modificação no próprio Linux – o núcleo do sistema operacional – a implementação deverá ser submetida para inclusão na distribuição oficial do *kernel*. Entre outros fatores, a inclusão estará sujeita à aceitação da comunidade da ideia conceitual, e da estabilidade do código previamente assegurada por testadores voluntários.

Partindo da implementação de referência, qualquer sistema operacional baseado em Linux poderia facilmente suportar a abordagem. Até mesmo o Android, que é um sistema operacional móvel, mas mesmo assim sujeito às tão comuns redes que operam via NAT.

Para tornar a implementação mais amigável, poderia ser criada uma interface de usuário gráfica para o *daemon*, integrada ao *shell* do sistema. Se feito como uma aplicação independente, que se comunica com o *daemon*, cada sistema poderia ter sua própria interface gráfica, que se encaixaria melhor no seu ambiente de usuário. No Ubuntu, por exemplo, tal interface poderia ser uma extensão da aplicação NetworkManager, que é por padrão visível na área de trabalho do usuário e já permite monitorar e configurar as atividades da rede. Através desta interface, o usuário deveria ser capaz de monitorar o *status* das portas automaticamente redirecionadas no roteador.

5.1.1 Interface Virtual para a Internet

Para além da implementação em si, a própria abordagem ainda pode ser melhorada, aumentando ainda mais a abstração. As interfaces de rede dos sistemas operacionais já são entidades abstratas por si só, podendo tanto serem um dispositivo físico quanto um dispositivo virtual, como as interfaces de *loopback*, PPP[1], VPNs[2] e túneis para outras redes. Considerando este fato, a própria conexão com a Internet poderia ser dada como uma nova interface de rede virtual.

A nova interface teria como endereço IP o endereço público do *gateway*, e as tentativas de escutar por uma de suas portas ativaria o redirecionamento automático. Como a conexão com a Internet apareceria para as aplicações como sendo uma interface local, qualquer aplicação que supõe uma conexão direta com a Internet funcionaria naturalmente, inclusive as que, para seus fins, precisam comunicar endereço IP e portas aos pares remotos, que é o caso citado na Seção 3.2.3, e que a proposta atual falha em resolver.

5.1.2 Futuro

Finalmente, a Internet vem sendo construída incrementalmente, são pequenos blocos de técnica e conhecimento adicionados sobre tudo o que já existe. Afinal, não se trata de ter a solução mais eficiente ou tecnicamente melhor, mas sim de estar acessível e

[1] Protocolo Ponto-a-Ponto.
[2] *Virtual Private Networks*.

conectado. De nada adianta o novo protocolo de Internet se ninguém mais usá-lo, ou um transporte mais seguro, se ninguém mais puder entender as mensagens, ou ainda uma novíssima pilha completa com uma outra Internet, se as aplicações não puderem utilizá-la. Eventualmente, imagina-se que a Internet tornar-se-á algo completamente diferente do que é hoje, mas isto não poderá ser feito ignorando-se o que já existe. Portanto, a interface POSIX de *sockets* ainda deve ser a chave, o denominador comum, entre a Internet atual e qualquer tentativa relevante de avançá-la.

Referências

[Argaez 2011] Argaez, E. D. (2011). World Internet Usage and Population Statistics. `http://www.internetworldstats.com/stats.htm`, acesso em fevereiro de 2012.

[Belimpasakis et al. 2007] Belimpasakis, P., Saaranen, A., e Walsh, R. (2007). Home DNS: Experiences with Seamless Remote Access to Home Services. In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, pp. 1 –8.

[Bernard 2011] Bernard, T. (2011). MiniUPnP Project HomePage. `http://miniupnp.free.fr/`, acesso em dezembro 2011.

[Bowden et al. 2009] Bowden, T., Bauer, B., Nerin, J., Feng, S., e Seibold, S. (2009). The `/proc` filesystem. `Documentation/filesystems/proc.txt`. Contido na distribuição em código fonte do Linux.

[Braden 1989a] Braden, R. (1989a). Requirements for Internet Hosts - Application and Support. RFC 1123 (Standard). Updated by RFCs 1349, 2181, 5321, 5966.

[Braden 1989b] Braden, R. (1989b). Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard). Updated by RFCs 1349, 4379, 5884, 6093, 6298.

[Cheshire et al. 2008] Cheshire, S., Krochmal, M., e Sekar, K. (2008). NAT Port Mapping Protocol (NAT-PMP). Internet-Draft.

[Colitti et al. 2010] Colitti, L., Gunderson, S. H., Kline, E., e Refice, T. (2010). Evaluating IPv6 adoption in the internet. In *Proceedings of the 11th international conference on Passive and active measurement*, PAM'10, pp. 141–150, Zurich, Switzerland, Berlin, Heidelberg. Springer-Verlag.

[de Souza Pereira et al. 2009] de Souza Pereira, J. H., Kofuji, S. T., e Rosa, P. F. (2009). Horizontal Address Ontology in Internet Architecture. In *New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on*, pp. 1 –6.

[Deering e Hinden 1998] Deering, S. e Hinden, R. (1998). Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard). Updated by RFCs 5095, 5722, 5871.

[Ford et al. 2005] Ford, B., Srisuresh, P., e Kegel, D. (2005). Peer-to-peer communication across network address translators. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pp. 13–13, Anaheim, CA, Berkeley, CA, USA. USENIX Association.

[Frex 2012] Frex (2012). Universal Plug'n'Play – OpenWrt Wiki. `http://wiki.openwrt.org/doc/howto/upnp#security.considerations`, acesso em fevereiro de 2012.

[FSF 2012] FSF (2012). GNU Licenses. `http://www.gnu.org/licenses/licenses.en.html`, acesso em junho de 2012.

[George et al. 2007] George, G., Wirt, E., e Blueman, D. J. (2007). Linux UPnP Internet Gateway Device – Documentation. `http://linux-igd.sourceforge.net/documentation.php`, acesso em dezembro 2011.

[Google 2012] Google (2012). IPv6 Statistics. `http://www.google.com/intl/en/ipv6/statistics/`, acesso em fevereiro de 2012.

[Haber et al. 2009] Haber, A., De Mier, J., e Reichert, F. (2009). Virtualization of Remote Devices and Services in Residential Networks. In *Next Generation Mobile Applications, Services and Technologies, 2009. NGMAST '09. Third International Conference on*, pp. 182 –186.

[Hemel 2006] Hemel, A. (2006). Universal Plug and Play: Dead simple or simply deadly? In *System Administration Network Engineering, 2006*.

[Huang et al. 2010] Huang, T.-C., Zeadally, S., Chilamkurti, N., e Shieh, C.-K. (2010). A programmable network address translator: Design, implementation, and performance. *ACM Trans. Internet Technol.*, 10(1):3:1–3:37.

[IANA 2011] IANA (2011). IPv4 Address Space Registry. `http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml`, acesso em fevereiro de 2012.

[ISO/IEC 1994] ISO/IEC (1994). *7498-1, Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*.

[Linux Foundation 2009] Linux Foundation (2009). generic_netlink_howto. `http://www.linuxfoundation.org/collaborate/workgroups/networking/generic_netlink_howto`, acesso em fevereiro de 2012.

[Mochalski e Schulze 2009] Mochalski, K. e Schulze, H. (2009). Ipoque Internet Study 2008/2009. Technical report, Ipoque.

[Mochel e Murphy 2010] Mochel, P. e Murphy, M. (2010). sysfs – The filesystem for exporting kernel objects. `Documentation/filesystems/sysfs.txt`. Contido na distribuição em código fonte do Linux.

[Muller et al. 2008] Muller, A., Carle, G., e Klenk, A. (2008). Behavior and classification of NAT devices and implications for NAT traversal. *Network, IEEE*, 22(5):14 –19.

[Norberg 2010] Norberg, A. (2010). μTorrent Transport Protocol. `http://bittorrent.org/beps/bep_0029.html`, acesso em fevereiro de 2012.

[Postel 1980] Postel, J. (1980). User Datagram Protocol. RFC 768 (Standard).

[Postel 1981] Postel, J. (1981). Transmission Control Protocol. RFC 793 (Standard). Updated by RFCs 1122, 3168, 6093.

[Rekhter et al. 1994] Rekhter, Y., Moskowitz, B., Karrenberg, D., e de Groot, G. (1994). Address Allocation for Private Internets. RFC 1597 (Informational). Obsoleted by RFC 1918.

[Rekhter et al. 1996] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., e Lear, E. (1996). Address Allocation for Private Internets. RFC 1918 (Best Current Practice).

[Seah et al. 2009] Seah, D., Leong, W. K., Yang, Q., Leong, B., e Razeen, A. (2009). Peer NAT proxies for peer-to-peer games. In *Proceedings of the 8th Annual Workshop on Network and Systems Support for Games*, NetGames '09, pp. 6:1–6:6, Paris, France, Piscataway, NJ, USA. IEEE Press.

[Smith e Lipner 2011] Smith, L. e Lipner, I. (2011). Free Pool of IPv4 Address Space Depleted. `http://www.nro.net/news/ipv4-free-pool-depleted`, acesso em fevereiro de 2012.

[Srisuresh e Egevang 2001] Srisuresh, P. e Egevang, K. (2001). Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational).

[UPnP Forum 2008] UPnP Forum (2008). UPnP Device Architecture. `http://upnp.org/sdcps-and-certification/standards/device-architecture-documents/`, acesso em fevereiro de 2012.

[UPnP Forum 2010] UPnP Forum (2010). UPnP IGD WANIPConnection. `http://upnp.org/specs/gw/igd2/`, acesso em dezembro de 2011.

[Vella et al. 2012] Vella, L. C., Camargos, L. J., e Rosa, P. F. (2012). A Complementary Approach for Transparent NAT Connectivity. In *ICN 2012, The Eleventh International Conference on Networks*.